UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

CONSOLIDAÇÃO E OPTIMIZAÇÃO DA MONITORIZAÇÃO E ALARMÍSTICA DE SISTEMAS DA ASSOCIAÇÃO DNS.PT

André Manuel Amaro Matias

PROJETO

MESTRADO EM ENGENHARIA INFORMÁTICA
Arquiteturas, Sistemas e Redes de Computadores

2015

UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

CONSOLIDAÇÃO E OPTIMIZAÇÃO DA MONITORIZAÇÃO E ALARMÍSTICA DE SISTEMAS DA ASSOCIAÇÃO DNS.PT

André Manuel Amaro Matias

PROJETO

MESTRADO EM ENGENHARIA INFORMÁTICA

Arquiteturas, Sistemas e Redes de Computadores

Trabalho orientado pelo Prof. Doutor Carlos Eduardo Ramos dos Santos Lourenço

e co-orientado por Assis Guerreiro

2015

Agradecimentos

Este projeto seria inconcebível sem a preciosa ajuda de toda a Associação DNS.PT, com especial ênfase na ajuda prestada pela área técnica, cujos conhecimentos permitiram a concretização de vários aspetos deste projeto, especialmente no que diz respeito ao sistema de monitorização e alarmística.

Quero também formalmente agradecer a todos os colaboradores da Associação, transversalmente a todas as áreas, cuja ajuda, disponibilidade e amabilidade facilitaram a integração na equipa e tornaram a estadia nas instalações da Associação mais agradável. À Ana Cunha, Isabel Miranda, Sónia Veloso, Sandra Costa, Joana Pena, Sara Monteiro Assis Guerreiro. Eduardo Duarte, Pedro Goes, Luísa Gueifão, Inês Esteves e Marta Moreira, deixo um especial apreço por todo o apoio prestado.

Quero também agradecer a disponibilidade que os intervenientes da parte da Faculdade de Ciências da Universidade de Lisboa tiveram neste projeto, a sua ajuda foi importante para que este projeto chegasse a bom-porto. Ao Professor Doutor Carlos Lourenço um obrigado pelo seu trabalho na orientação desta dissertação.

Por fim quero agradecer a todos os amigos e familiares, cujo apoio em momentos mais difíceis aliviaram a carga que este projeto apresentou durante nove meses. O constante apoio, presença e conselho que prestaram foram fundamentais para o sucesso deste projeto. Deixo então um especial agradecimento a Manuel, Isabel, Christine e Lucie Matias, à Carolina Carvalho, ao Flávio Saraiva, Eduardo Matos, Ruben Campos e Hugo Sousa, entre muitos outros, sem os quais não seria hoje a mesma pessoa.

Resumo

A Associação DNS.PT, na qualidade de responsável pela gestão, registo e manutenção de domínios sob o Domínio de Topo Português (.pt), gere um serviço fundamental para o bom funcionamento da Internet Portuguesa. Este serviço necessita de constante monitorização quanto aos recursos utilizados, para garantir a disponibilidade e correto funcionamento. Neste contexto, é necessário que exista um sistema que recolha dados em tempo-real para monitorizar os recursos afetos ao serviço e que, caso existam indicadores que apresentem condições preocupantes, se despoletem os mecanismos certos para alarmística.

O projeto apresenta um conjunto de tarefas de análise e crítica ao anterior sistema, permitindo estabelecer uma base para a criação de monitorização e alarmística que se adapte à realidade da Associação. Para concretizar o novo sistema de monitorização e alarmística, executou-se um conjunto de tarefas de implementação, com iterações de levantamento de requisitos, desenho e avaliação da solução, havendo sempre a preocupação de manter o útil e repensar o obsoleto.

Numa primeira fase (de Setembro a Dezembro de 2013), fez-se o levantamento para perceber extensivamente quais os problemas a abordar, o que existia e o que precisava de ser mudado. Procedeu-se, também, à implementação de sistemas de monitorização auxiliar, como o DSC.

A fase final do projeto (de Janeiro a Julho de 2014) consistiu em três etapas distintas: planeamento e desenho do novo sistema de monitorização e alarmística, implementação da solução, e criação de uma interface gráfica para esta. Esta última teve especial relevancia devido à importância que a apresentação dos dados representa para a Associação DNS.PT.

Palavras-chave: Monitorização, Alarmística, Associação DNS.PT, *Domain Name System*

Abstract

The Associação DNS.PT, in its role of manager, registry, and maintainer of the domains under the Portuguese Top Level Domain (.pt), manages a service fundamental to the well-being of the Portuguese Internet, and so, has the need for constant monitoring regarding the resources used to assure the availability of said service. In this context, it is vital that there is a system gathering data in real-time to monitor the resources tied to the services that the institution offers, and that, in the event of concerning values, the right alerting mechanisms are triggered.

The project presents a set of tasks of analysis and criticism to the previous system, establishing a starting point for the creation of monitoring and alarming suited to the reality of the Associação, as well as a set of tasks for implementation of an information system, with iterations such as research, design and evaluation, in order to materialize the new monitoring and alarming system, always doing the best effort to maintain what was useful and rethink what was obsolete.

In the first phase (from September through December of 2013), research was made to extensively understand what were the problems to address, what existed and what needed changing. During this time, implementation of auxiliary monitoring systems, such as the DSC, also took place.

The final phase of this project (from January through July of 2014) consisted in three diferent stages: planning and design of the new monitoring and alarming system, implementation of the new system, and the creation of an interface for said system. This last stage assumed special relevance, because of the importance of how the data is presented to the Associação DNS.PT.

Keywords: Monitoring, Alerting, *Associação DNS.PT*, Domain Name System

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O DNS (sigla para *Domain Name System*, ou Sistema de Nomes de Domínio) é um sistema de bases de dados distribuído num esquema de comunicação cliente-servidor [1]. Cada fragmento desta base de dados é responsável por uma fração de informação acerca do ponto onde se encontra e de como chegar a outros fragmentos da base de dados, onde se pode encontrar mais informação. A distribuição apresenta-se no formato de uma hierarquia em árvore com uma raiz[1] bem conhecida, de onde todos os nós provêm, diretamente ou por descendência de nós que ligam diretamente. Esta base de dados contém informação sobre os recursos da Internet, que necessitam de uma referência para serem conhecidos: uma mnemónica (domínio) e um identificador inequívoco (um endereço IP) [1].

Um domínio de DNS é um nome que serve para para localizar e identificar, de forma unívoca, computadores na Internet. É um conjunto de informações necessárias para se poder chegar ao servidor pretendido (para aceder ao um *website, e-mail* e outros serviços que estejam disponibilizados através da Internet).

Cada recurso na Internet está disponível de duas formas: ou por nome – nome legível e com significado literal – ou por endereço IP – código numérico com significado para as máquinas [1]. O ser humano compreende e retém melhor a informação sob a forma de palavras (mnemónicas) do que números, e os computadores o contrário. É, então, necessário encontrar um tradutor entre as duas "linguagens", um sistema que faça a tradução de forma transparente ao utilizador da Internet para que este chegue aos recursos desejados. Esta necessidade foi o catalisador que impulsionou a génese do DNS.

O DNS aparenta ser um sistema relativamente simples e intuitivo: guarda-se a informação para localizar os serviços que são disponibilizados na Internet e acede-se a esse "repositório" para se saber a tradução do nome para o IP respetivo do recurso em

[1] Servidores raiz ou *root-servers* são servidores que se encontram no topo da hierarquia, com endereços IP estáticos, que servem de ponto inicial para a pesquisa na hierarquia de nomes.

causa, ou vice-versa. No entanto, uma das premissas do DNS é de ser distribuído, pelo que a noção "repositório" – algo centralizado – não é viável.

A permissa da distribuição do DNS leva a que seja necessário espalhar as informações deste por vários servidores – servidores de nomes, responsáveis por traduzir nomes para endereços IP e vice-versa – e criar uma hierarquia que permita, de qualquer ponto da rede, obter informação necessária para aceder ao recurso que se pretende. Esta abordagem leva a que seja necessário lidar com o problema da colisão de nomes de domínios, que surge quando se assume uma dimensão muito grande para um espaço de nomes finito e onde as mnemónicas podem facilmente repetir-se [1].

Este esquema hierarquico resolve vários problemas, como por exemplo, a tradução de um endereço IP ou de um nome de domínio, ou a tentativa de ter domínios iguais (ao impor uma camada de gestão na hierarquia). Com a introdução de *root-servers*, resolve-se o problema de saber por onde começar a procura, introduzindo o ponto de partida para a pesquisa. Estes servidores de nomes sabem quem são os servidores responsáveis por domínios diretamente sob eles, domínios apelidados de "topo", que são as primeiras "folhas" nesta estrutura.

Os domínios de topo (*Top Level Domains*) dividem-se em duas categorias: ccTLD (*Country Code* TLD ou Domínio de Topo de Código de País) e gTLD (*Generic* TLD ou Domínio de Topo Genérico), onde o primeiro se refere a domínios afetos a países e o segundo a domínios de cariz geral, sem ligação direta a um país. A Associação DNS.PT, na qualidade de ccTLD para Portugal, é o *Registry* do .pt ou seja, é responsável pela gestão, registo e manutenção da zona .pt, que é o conjunto dos domínios associados a Portugal (na grande maioria). Os ccTLD são responsáveis por providenciar as informações relevantes para aceder a domínios sob o .<*país*> e, no caso do *ccTLD* Português, responsável por disponibilizar informação sobre os domínios .pt.

No entanto, a cadeia de resolução de nomes não termina aqui. Uma camada abaixo, encontramos, de novo, servidores de nomes, responsáveis pela resolução de nomes dos subdomínios do domínio .pt (*dns.pt* ou *ul.pt*, por exemplo). Neste panorama, entram os *Registrars* ou Agentes de Registo, entidades intervenientes no processo de resolução, ao providenciar o contacto com o utilizador final (um cliente de cariz doméstico ou empresarial) que não fornece serviço de resolução de nomes para outros utilizadores, ou seja, que é o nó final da cadeia de resolução de nomes. O cc*TLD* Português é então responsável por oferecer o serviço de resolução de nomes para identificar o servidor de nomes destes *Registrars*, entidades que contactam diretamente com a Associação DNS.PT e que são o *core* de negócio da instituição.

É importante ter em consideração que alguns *Registries* também podem ser *Registrars*, no sentido em que podem eles próprios aceitar o registo de domínios e

delegar os mesmos. Os *Registrars* necessitam, ainda, de ser acreditados por um *Registry* para que possam exercer as funções de delegação e gestão de nomes de domínios que lhes pertençam, pelo que se garante que a cadeia de resolução é sempre gerida pelo cc*TLD* com consciência dos intervenientes no processo de resolução de nomes. Os *Registrars* podem registar domínios diretamente sob o cc*TLD* (por exemplo <*nome*>.*pt*) ou configurar domínios sob um subdomínio exclusivamente seu (como por exemplo <*nome*>.<*subdomínio*>.*pt*).

Sumariamente, para resolver um nome de domínio em .pt, um certo cliente terá de percorrer a hierarquia, começando num *root-server*, passando para o servidor de nomes .pt, passando depois pelo servidor de nomes de um *Registrar* com o endereço IP do servidor que contém o recurso que se procura.

É, então, possível perceber a importância da Associação DNS.PT na "saúde" da Internet Portuguesa e a influência na Internet em geral: sem servidor de nomes para a zona .pt não existe forma de aceder aos recursos (atualizados) que estão disponíveis ao mundo sob o domínio .pt. Sem *Resgistry* funcional, toda a lógica de negócio associada com o registo de domínios é posta em causa, e os *Registrars* deixam de poder exercer as suas funções corretamente.

Mas o protocolo DNS é apenas um dos pontos fulcrais onde este projeto vai incidir. Esta contextualização pretende ilustrar a quantidade de sistemas que o DNS compreende e que terão de ser monitorizados. Existe um conjunto de outros sistemas (sistema SIGA) que suportam a restante atividade que foi desenvolvida em redor do serviço de *Registry*, como serviços *Web* que são prestados ao público e clientes da Associação, existindo, ainda, sistemas de bases de dados que albergam informações críticas não só para o sistema de DNS, mas também para os outros sistemas que foram criados no seguimento das funções da Associação DNS.PT. Todos estes sistemas necessitam de ser constantemente monitorizados, pois o sucesso da atividade profissional da Associação DNS.PT depende do bom funcionamento de todos eles.

1.1 Motivação

A Associação DNS.PT tem responsabilidades quanto à manutenção de uma das infra-estruturas que suporta a Internet Portuguesa. Caso haja interrupção do serviço que leve à total paragem da infra-estrutura, a estabilidade e disponibilidade da Internet Portuguesa é posta em causa. Embora seja um cenário *in extremis*, é possível ocorrer e, como tal, tem de ser contemplado e tratado com antecedência, para que nunca ocorra.

Neste aspeto, as principais fraquezas que o antigo sistema de monitorização apresenta são:

1) Capacidade de saber, em tempo-real, a quantidade de recursos afetos aos servidores de nomes:

a) Quanto a recursos das máquinas físicas (CPU, memória, disco);

b) Funcionamento do *BIND*, quanto às perguntas que estão a chegar e as respostas a serem dadas.

2) Severas lacunas, quer na monitorização, quer na alarmística do sistema SIGA, quanto a:

a) *Web Services*

i) Disponibilidade

ii) Tempos de Resposta

b) *JVM*

i) Funcionamento

c) Bases de dados

i) Estado

ii) Volumes de pedidos (SELECT, UPDATE, etc.)

3) Obtenção de dados insuficientes para a monitorização compreensiva das máquinas e outros recursos físicos/virtuais da Associação.

4) Forma de apresentação dos dados de monitorização e de alarmística.

1.2 Objetivos

Os objetivos deste trabalho são claros: com base na atual implementação da monitorização e alarmística da Associação DNS.PT, conceber uma solução que permita abranger, em termos de monitorização, todos os recursos vitais para a manutenção dos vários serviços que se prestam e configurar o sistema para que, aquando da existência de motivos de alarme, se despoletem os mecanismos certos para informar acerca da ocorrência.

Esse processo deve estar bem afinado para que se possam mitigar os efeitos nefastos e deve conseguir guardar registo dos eventos para efeitos de análise e

depuração das causas da ocorrência e de potenciais causadores, principalmente no caso de ataques premeditados.

Para concretizar o sistema que se pretende, são traçados os seguintes objectivos e respectivas formas de avaliação:

1) Análise crítica do antigo sistema.

– Será necessário avaliá-lo criteriosamente, pretendendo-se reunir com equipa para identificar problemas a tratar e virtudes a manter.

– Fazer levantamento do que precisa de ser revisto, de acordo com a qualidade da informação que presta, performance, relevância e competência dos processos.

– A avaliação será feita tendo em conta a validação pelos restantes elementos da equipa, bem como do resultado da investigação em fontes externas (recursos bibliográficos e *webgráficos*).

2) Desenho de um novo sistema e planeamento da implementação.

– Verificar se se tem as tecnologias necessárias para concretizar, bem como o conhecimento técnico para implementar.

– Considerar as conclusões retiradas da avaliação feita anteriormente para elaborar um esboço daquilo a que se quer chegar, identificando o que se deve manter, melhorar e implementar.

– Avaliar a proposta de acordo as opiniões da equipa técnica, com recurso ao que foi evidenciado pelos levantamentos.

3) Implementação da solução

– Plano de desenvolvimento flexível e baseado em iterações (método ágil de desenvolvimento de *software*), com etapa de avaliação para saber o que se pode melhorar, no fim de cada iteração.

– O *feedback* da equipa e os testes serão fulcrais para esta fase, pelo que se pretende reunir com a equipa e ter períodos de testes para avaliar a solução e encontrar motivos de melhoria.

4) Identificação de oportunidades de melhoria

– Identificar pontos de melhoria, nomeadamente, implementações que foram descartadas em detrimento de outras com mais urgência ou simplesmente novos requisitos.

– Os resultados obtidos da utilização do novo sistema, auxiliados pelas opiniões da equipa, deverão potenciar a identificação de oportunidades de melhoria.

1.3 Planeamento

O Planeamento pretende traçar uma calendarização mais ou menos fiel ao que se pretende ser o fluxo de trabalho durante os 9 meses de duração do projeto. As fases do trabalho seguem o formato padrão para um desenvolvimento de *software*: levantamento e análise de requisitos, desenvolvimento e implementação, fase de testes e produção de documentação [2]. A Tabela 1 apresenta a descrição detalhada das tarefas que constituem o trabalho, que resulta na elaboração de uma Mapa de *Gantt*, que está visível na Figura 1, com a calendarização e dependências das tarefas.

Task Name	Duration	Start	Finish	Pre
1 - Projecto Monitorização	**196 days**	**Mon 16-09-13**	**Mon 16-06-14**	
2 - Inserção no ambiente de trabalho	**6 days**	**Mon 16-09-13**	**Mon 23-09-13**	
3 - Reunião de arranque	1 day	Mon 16-09-13	Mon 16-09-13	
4 - Apresentação à equipa	2 days	Tue 17-09-13	Wed 18-09-13	3
5 - Apresentação dos recursos, instalações e serviços da organização	3 days	Thu 19-09-13	Mon 23-09-13	4
6 - Análise e especificação da solução	**66 days?**	**Tue 24-09-13**	**Tue 24-12-13**	**5**
7 - Análise prospectiva	**15 days**	**Tue 24-09-13**	**Mon 14-10-13**	**5**
8 - Análise prospectiva	2 days	Tue 24-09-13	Wed 25-09-13	5
9 - Levantamento de requisitos	13 days	Thu 26-09-13	Mon 14-10-13	8
10 - Análise funcional	**17 days**	**Tue 15-10-13**	**Wed 06-11-13**	**9**
11 - Definição de Objectivos	5 days	Tue 15-10-13	Mon 21-10-13	9
12 - Definição de itens de monitorização	10 days	Tue 22-10-13	Mon 04-11-13	11
13 - Definição de métricas de monitorização	2 days	Tue 05-11-13	Wed 06-11-13	12
14 - Análise não funcional	**6 days**	**Thu 07-11-13**	**Thu 14-11-13**	**13**
15 - Definição de Objectivos	3 days	Thu 07-11-13	Mon 11-11-13	13
16 - Definição de parametros não funcionais	1 day	Tue 12-11-13	Tue 12-11-13	15
17 - Definição de métricas de monitorização	1 day	Wed 13-11-13	Wed 13-11-13	16
18 - Reunião de controlo de projecto	1 day	Thu 14-11-13	Thu 14-11-13	17
19 - Análise técnica da solução	**18 days**	**Fri 15-11-13**	**Tue 10-12-13**	**18**
20 - Arquitectura	5 days	Fri 15-11-13	Thu 21-11-13	18
21 - Integração da monitorização nos serviços	9 days	Fri 22-11-13	Wed 04-12-13	20
22 - Configuração (Parametrizações)	4 days	Thu 05-12-13	Tue 10-12-13	21
23 - Análise de propostas de novas	**9 days**	**Wed 11-12-13**	**Mon 23-12-13**	**22**

tecnologias				
24 - Identificação	5 days	Wed 11-12-13	Tue 17-12-13	22
25 - Analise	2 days	Wed 18-12-13	Thu 19-12-13	24
26 - Comparação	1 day	Fri 20-12-13	Fri 20-12-13	25
27 - Reunião de controlo de projecto	1 day	Mon 23-12-13	Mon 23-12-13	26
28 - DELIVERABLES - Análise e especificação da solução	**1 day?**	**Tue 24-12-13**	**Tue 24-12-13**	**27**
29 - Requisitos da solução	1 day?	Tue 24-12-13	Tue 24-12-13	
30 - Especificação de solução	1 day	Tue 24-12-13	Tue 24-12-13	
31 - Preparação do ambiente de desenvolvimento	**41 days**	**Tue 24-12-13**	**Tue 18-02-14**	**27**
32 - Aquisição de Recursos	**31 days**	**Tue 24-12-13**	**Tue 04-02-14**	**27**
33 - Elaboração de caderno de especificação técnica	1 day	Tue 24-12-13	Tue 24-12-13	27
34 - Pedidos de cotação para equipamento	7 days	Wed 25-12-13	Thu 02-01-14	33
35 - Equipamento adjudicado	5 days	Fri 03-01-14	Thu 09-01-14	34
36 - Aquisição	18 days	Fri 10-01-14	Tue 04-02-14	35
37 - Preparação do ambiente	**10 days**	**Wed 05-02-14**	**Tue 18-02-14**	**32**
38 - Instalação do equipamento	3 days	Wed 05-02-14	Fri 07-02-14	36
39 - configuração de equipamento	2 days	Mon 10-02-14	Tue 11-02-14	38
40 - Instalação e configuração do ambiente (Software e Serviços)	5 days	Wed 12-02-14	Tue 18-02-14	39
41 - Implementação da solução	**68 days?**	**Wed 19-02-14**	**Fri 23-05-14**	**40**
42 - Instalação de sistemas	10 days	Wed 19-02-14	Tue 04-03-14	40
43 - Configuração de sistemas	10 days	Wed 19-02-14	Tue 04-03-14	40
44 - Integração entre os vários sistemas	**40 days**	**Wed 05-03-14**	**Tue 29-04-14**	**43**
45 - Implementação de novas funcionalidades	20 days	Wed 05-03-14	Tue 01-04-14	43
46 - Parametrizações	19 days	Wed 02-04-14	Mon 28-04-14	45
47 - Reunião de controlo de projecto	1 day	Tue 29-04-14	Tue 29-04-14	46
48 - Testes	**7 days?**	**Wed 30-04-14**	**Thu 08-05-14**	**47**
49 - Elaboração de plano de testes	1 day?	Wed 30-04-14	Wed 30-04-14	47
50 - Execução dos testes	5 days	Thu 01-05-14	Wed 07-05-14	49
51 - Produção de relatório de testes	1 day?	Thu 08-05-14	Thu 08-05-14	50
52 - Produção de documentação da solução	10 days	Fri 09-05-14	Thu 22-05-14	51
53 - DELIVERABLES - Documentação da solução	**1 day?**	**Fri 23-05-14**	**Fri 23-05-14**	**52**
54 - Manual de Utilização	1 day?	Fri 23-05-14	Fri 23-05-14	

55 - Listagem detalhada de monitorização	1 day?	Fri 23-05-14	Fri 23-05-14	
56 - Relatório de testes	1 day?	Fri 23-05-14	Fri 23-05-14	
57 - Reunião de controlo de projecto	1 day	Fri 23-05-14	Fri 23-05-14	52
58 - Entrada em produção e Operação	1 day	Mon 09-06-14	Mon 09-06-14	57
59 - Revisão da solução	**5 days**	**Tue 10-06-14**	**Mon 16-06-14**	**58**
60 - Refinamento da solução (Tunning)	2 days	Tue 10-06-14	Wed 11-06-14	58
61 - Identificar trabalho Futuro	2 days	Wed 11-06-14	Thu 12-06-14	58
62 - Reunião de controlo de projecto (final)	1 day	Fri 13-06-14	Fri 13-06-14	58
63 - Entrega de Projeto - Fim	0 days	Mon 16-06-14	Mon 16-06-14	62

Tabela 1 – Planeamento de Tarefas

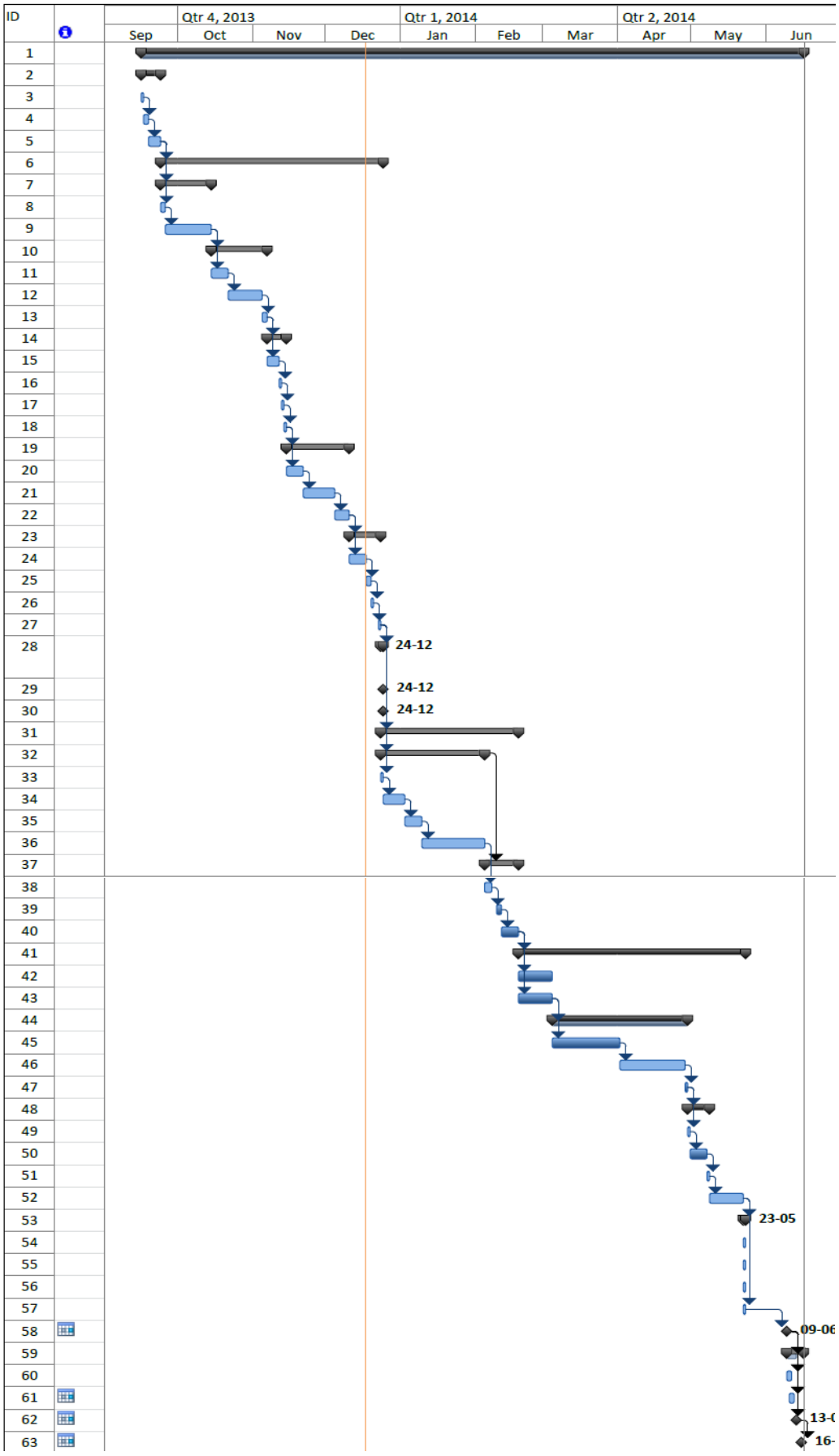


Figura 1 - Mapa de *Gantt*

1.4 Organização do documento

A organização deste documento assenta na seguinte estrutura:

- Capítulo 2 - "Fase Preliminar – Antigo Sistema de Monitorização"

Faz-se o levantamento dos componentes do antigo sistema de monitorização, apresentam-se algumas tecnologias e metodologias que se usavam, e refere-se algum do trabalho que foi feito nos três primeiros meses do projeto, aquando da redação do relatório preliminar.

- Capítulo 3 - "Fase Final – Novo Sistema de Monitorização"

Explica todo o processo por detrás da idealização e conceção do novo sistema de monitorização, apresentando pormenorizadamente todos os detalhes do sistema de monitorização criado e o porquê de serem pensados dessa forma.

- Capítulo 4 - "Concretização do Novo Sistema de Monitorização"

Descreve toda a implementação que foi feita para concretizar o sistema de monitorização idealizado, desde a configuração do servidor *Zabbix* até à implementação de tecnologias para recolha de indicadores, configuração de alarmística e integração no sistema de monitorização.

- Capítulo 5 - "*Dashboard* DNS.PT"

Cobre todo o trabalho feito no âmbito da nova interface gráfica para o sistema de monitorização na Associação DNS.PT.

- Capítulo 6 - "Discussão"

É um capítulo de opinião pessoal, onde irei refletir sobre o balanço do projeto, como correu face ao planeamento, e perspetivas de trabalho futuro.

Capítulo 2

Fase Preliminar – Antigo Sistema de Monitorização

Desde a data de início de funções (16 de Setembro de 2013) até ao fim do ano 2013 (sensivelmente até meados de Dezembro de 2013), foram realizadas várias atividades no âmbito do conhecimento do panorama atual do sistema de monitorização e alarmística, na pesquisa e levantamento de requisitos a cumprir na conceção do novo sistema, e implementação de novas metodologias de monitorização da utilização dos recursos da Associação DNS.PT. Além de tudo isto, houve um momento de integração no contexto da Associação, na equipa e na missão desta.

2.1 Antigo sistema *Zabbix*

O *software Zabbix* é uma ferramenta robusta de monitorização de nível empresarial, que permite monitorizar em tempo-real a disponibilidade e performance dos recursos da organização, e capaz de representar os dados de monitorização graficamente [3]. Esta ferramenta apresenta uma interface gráfica *Web* que permite o acesso e configuração da monitorização e alarmística [3].

O *Zabbix* permite contacto com as máquinas e serviços que monitoriza através de protocolos como SNMP [11] (*Simple Network Management Protocol*, um *standard* na comunicação entre agentes e sistemas de gestão e obtenção dos indicadores que os agentes recolhem), HTTP (*HyperText Transfer Protocol*, forma mais usual de comunicação de dados na *Web*), SSH (*Secure Shell*, uma forma criptograficamente segura de controlo remoto de uma linha de comandos), e até por IPMI (*Intelligent Platform Management Interface*, serviços de acesso *out-of-band* ou seja, serviços a usar aquando da interrupção da comunicação com o *hardware*) para remotamente aceder a um sistema computacional e obter indicadores [3]. A forma mais usada na monitorização são os agentes *Zabbix*, que são pequenos serviços executados nas máquinas alvo que intercomunicam com o servidor *Zabbix* para variadas tarefas de monitorização.

Na questão do sistema *Zabbix*, o levantamento efetuado permitiu conhecer o presente da monitorização e alarmística da Associação, identificando-se:

- *Hosts*: Sistemas computacionais que funcionam numa rede IP (como servidores e outros dispositivos monitorizáveis) que hospedam serviços e que produzem informação (ou pelo menos permitem o acesso à informação) para o sistema de monitorização recolher [4];

- *Items*: São os indicadores (métricas de utilização de certos recursos) a recolher. Um *Item* está diretamente relacionado com a monitorização de um certo recurso numa máquina ou num conjunto delas [4].

- *Triggers*: É a conciliação entre a monitorização e a alarmística. Um *Trigger* associa indicadores (*Items*) com valores marginais para o qual se pretende realizar uma alteração de estado de monitorização de um recurso, caso o valor marginal não seja atingido, caso seja igualado ou caso seja ultrapassado. Um *Trigger* é o despoletar da ação primária de alarmística, ao fornecer informação de alarme sobre o estado de certo recurso [4].

- *Actions*: As ações ou *Actions* são a outra componente informativa da alarmística do *Zabbix* e comportam grande parte da alarmística do sistema vigente. Cada ação associa um ou vários *Triggers* a um procedimento [4]. Esse procedimento pode ser meramente informativo – um *e-mail*, um SMS, um *ticket* – ou pode ser reativo, realizando alguma ação para lidar com a ocorrência que despoletou o alarme – desligar ou ligar um sistema, bloquear ou desbloquear recursos, entre outros.

2.2 *Zabbix* no contexto da Associação DNS.PT

As configurações *Zabbix* padrão vão desde a monitorização do estado das máquinas (cargas de CPU, de memória, de disco, de rede) até ao estado de serviços (estados das bases de dados de ambientes de produção e qualidade, velocidades de resposta a *queries*; estados dos servidores de conteúdos – servidores *Apache* – quer de *frontend*, quer de *backoffice*; entre outros). Esta configuração permite obter um grande volume de dados, posteriormente usados para gerar gráficos que representam o que está a acontecer nos vários componentes do sistema.

Com o conhecimento das funções básicas do *Zabbix* e das suas utilizações na monitorização dos sistemas, foi importante perceber as outras funções mais avançadas usadas nesta linha. Foram identificadas as seguintes funcionalidades avançadas:

- ***BIND***: O *BIND* é o *software* que permite a resolução de nomes de domínio e a publicação destes para os outros sistemas computacionais na Internet [1]. É monitorizado recorrendo-se à utilização do *rndc stats*, uma funcionalidade de estatísticas do *BIND*, permitindo saber que *queries/requests* chegam aos servidores de nomes e que *replies* estes devolvem, entre outros indicadores. Na secção Anexo pode-se ver com detalhe, na Figura I, como se faz a monitorização no sistema atual.

- ***ZoneCheck***: o *ZoneCheck* é um script criado à medida das necessidades técnicas, para recolher os dados dos tempos de resposta dos servidores de nomes para a zona .pt, primário e secundários (estando estes dispersos geograficamente por vários países). A Figura II da secção Anexo é ilustrativa da monitorização feita.

- **NETSTAT**: Embora o *Zabbix* tenha a capacidade de obter indicadores para os dispositivos de rede, foram criados *scripts* que fazem a recolha de indicadores de rede baseando-se nas capacidades dos agentes do *Zabbix*. Estes agentes – *daemons* ou serviços autónomos em segundo plano – são configurados para executarem *scripts* que refinam a capacidade de obtenção normal que o *Zabbix* tem para indicadores de rede. Na Figura III da secção Anexo, pode-se ver a representação esquemática do funcionamento destes agentes para obtenção de dados de rede.

- **ORABBIX**: O ORABBIX [5] é uma componente externa que integra com o *Zabbix* para permite monitorizar as bases de dados *Oracle*. Esta componente fornece as informações das bases de dados que albergam informações de negócio, indicador importante devido à criticidade que as bases de dados têm no serviço de *Registry*. O esquema de funcionamento está graficamente disponível na Figura IV da secção Anexo.

- ***Web Items*/*Pingdom***: Forma de monitorização do SIGA (Sistema de Informação e Gestão Administrativa) – muito utilizado pelos colaboradores, e cujos serviços *Web* são essenciais para o exercício das funções técnicas, administrativas e de gestão de um ccTLD. É fulcral que o sistema seja constantemente monitorizado quanto à coerência das suas funcionalidades, algo que é conseguido utilizando, por parte do *Zabbix*, as funcionalidades de verificação de conteúdo *Web* – os *Web Items* – que assenta num outro sistema que disponibiliza os dados para a monitorização – o *Jenkins* [6].

O *Jenkins* não é, só por si, uma ferramenta de monitorização, mas sim uma fonte de indicadores que "alimenta" a monitorização. Esta ferramenta atua como um escalonador de *jobs* (ações que se devem executar com uma certa periocidade), executando um *URL* (um recurso *Web* que traduz uma funcionalidade do serviço) e devolvendo o seu valor de retorno (guardando um histórico de execução de *jobs*) [6]. A conjugação *Web Items* + *Jenkins* permite monitorização da coerência das funcionalidades de um *Web Service*, ao verificar se o valor de retorno dessas funcionalidades está de acordo com o esperado.

Para integrar com o servidor *Zabbix*, o *Jenkins* fornece os indicadores a monitorizar para que estes sejam filtrados por um código de retorno (por exemplo, um código "*HTTP 200 OK"*, ou um valor próprio do *Web Service* como *SUCCESS*).

No entanto, a monitorização externa dos conteúdos *Web* está a cargo do *Pingdom*, serviço externo à Associação DNS.PT, que monitoriza as respostas HTTP que se recebem aquando do acesso ao *Website www.dns.pt*. Caso o retorno HTTP esteja fora dos valores esperados, esta ferramenta ativa os mecanismos de alarmística, via SMS. O esquema de funcionamento dos *Web Items* está explicado na Figura V da secção Anexo.

Para compreender o panorama de monitorização da Associação DNS.PT, foi necessário perceber o que era efetivamente monitorizado – que serviços e servidores/dispositivos – e em que medida os indicadores estavam a ser utilizados para a alarmística. Neste âmbito, foi feito um levantamento pormenorizado, tão completo quanto possível, dos indicadores e valores marginais para despoletar a alarmística definida para cada servidor e serviço. Na secção Anexo consta a Tabela II com um excerto do levantamento feito.

2.3 A Monitorização na Associação DNS.PT

A monitorização feita utilizando o *Zabbix* baseia-se na leitura dos valores que vão sendo recebidos e na interpretação dos gráficos associados. A Figura 2 demonstra a utilização de um ecrã gigante para visualizar a informação fornecida pelo *Zabbix*.

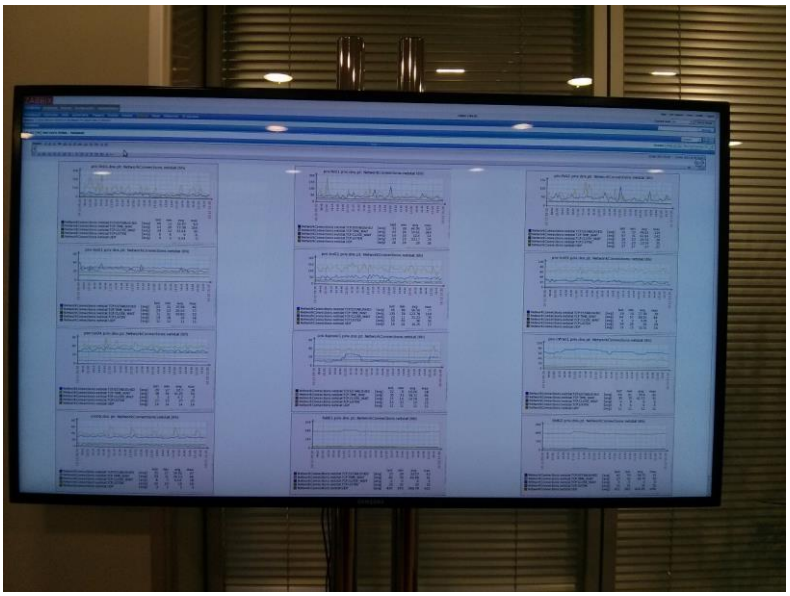


Figura 2 - Fotografia do Ecrã de Monitorização no escritório da Associação DNS.PT

Existem três servidores de monitorização, cada um específico para cada vertente de monitorização: um servidor que monitoriza os servidores e dispositivos quanto a indicadores de sistema operativo (métricas de desempenho, de rede, de armazenamento, etc.), um servidor que monitoriza os indicadores de aplicações (bases de dados, servidores *Web*, serviço de nomes, etc.) e um servidor que agrega esta informação para a geração de gráficos.

Esta separação permite dividir o domínio de monitorização em subdomínios mais específicos, facilitando o acesso ao tipo de informação pretendida. A figura abaixo mostra o monitor que é usado no escritório da Associação DNS.PT com vários gráficos em tempo-real, para consulta dos estados dos vários sistemas monitorizados.

2.4 DSC: Monitorização dos Servidores de Nomes

Uma vez finalizado o processo de conhecimento e ambientação ao sistema de monitorização e alarmística da Associação DNS.PT, passou-se à fase de investigação com o objetivo de solucionar os problemas do atual sistema.

Como primeiro projeto, a tarefa foi implementar o sistema de monitorização de servidores de nomes DSC (*DNS Statistics Collector*), do DNS OARC (*DNS Operations, Analysis, and Research Center*) [7], para se obter uma outra perspetiva de monitorização de servidores de nomes, diferente daquela que o *Zabbix* fornece. Por ser um *software* criado com o único propósito de monitorizar o DNS, o DSC oferece dados para monitorizar o tráfego DNS nos servidores de nomes.

Este *software*, que atualmente está vigente em todos os servidores de nomes geridos pela Associação DNS.PT, está dividido em duas componentes: um coletor de dados (*collector*) e um apresentador de dados (*presenter*).

2.4.1 Estrutura e Funcionamento do Collector

O *collector* é responsável por obter os dados do servidor de nomes em que está a correr, capturando e guardando os pacotes de rede usando o *lib_pcap*, uma biblioteca de funções que permitem capturar os pacotes TCP/IP que passam numa dada interface de rede. O *collector* condensa as informações nesses pacotes em ficheiros XML (*eXtensible Markup Language*), cuja estrutura está associada aos *datasets* (conjuntos de dados a representar num gráfico) definidos no ficheiro de configuração [7].

Cada *collector* tem uma diretoria onde se guardam os dados capturados (*/run*), dentro da qual existe a diretoria com o nome do servidor de nomes no contexto do

sistema. Os ficheiros XML que vão sendo criados ao longo do tempo são guardados na diretoria */run/<nome_ns>/upload.* Nesta agrupam-se os ficheiros XML por data na diretoria *<nome do presenter>* para que se possa enviar sempre os dados com consciência temporal. Um exemplo de caminho para encontrar os dados no servidor de nomes *ns.dns.pt* é: *dsc/run/ns.dns.pt/upload/presenter/2013-11-16/.*

A função da máquina *collector* no sistema DSC é simples de compreender: capturar os pacotes respetivos ao tráfego DNS e condensar essa informação em ficheiros XML; transferir os ficheiros para a máquina que os vai apresentar, através de mecanismo de transferência de dados remota – *rsync*, um sistema de transferência de dados incremental – usando SSH para comunicação segura; e *scripts* executados periodicamente para limpar os dados que já foram transmitidos. Os dados são transmitidos para uma outra máquina – o *presenter* – que tem o objectivo de os apresentar graficamente e cujo funcionamento é descrito de seguida.

2.4.2 Estrutura e Funcionamento do Presenter

A máquina *presenter*, ao contrário da máquina *collector*, não tem um programa a correr constantemente, pois funciona à base de *scripts* na linguagem de programação *Perl* (dos quais, o CGI usado para apresentar no *browser* a página interativa das estatísticas) que trabalham os dados de forma periódica [7]. O objectivo é transformar os dados que chegam em XML em gráficos que demonstram a utilização dos servidores de nomes.

A compreensão do esquema cliente-servidor que está subjacente à comunicação *collector-presenter* é conseguida através de encadeamento de diretorias que têm um significado próprio para a aplicação: no *collector*, cada diretoria de *presenter* indica para onde o *collector* irá enviar os seus dados, enquanto no *presenter* se tem o oposto, uma diretoria por cada *collector,* de onde se recebem os dados.

O *presenter*, à semelhança do *collector*, tem uma estrutura bem definida de diretorias para que consiga compreender de onde vêm os dados, nomeadamente, de que máquinas vêm os dados, num esquema semelhante ao *collector* em estrutura, mas com as seguintes diferenças:

Existe a diretoria */data* em vez de */run*, onde se encontram os dados vindos de outros *collector*s. Dentro desta diretoria encontra-se a diretoria com o nome do grupo de *collectors*, e é nesta que se definem de que *collectors* se recebem dados. Para cada *collector* tem de haver uma diretoria com o nome dele no sistema.

Dentro dessa, a estrutura é diferente da do *collector*: existe uma diretoria para cada dia, com dados para efeitos de janelas temporais, com extensão *.dat* para compressão; uma diretoria *done* com os dados em XML, para ser mais rápido o acesso à informação dos últimos 5 dias (este parâmetro é personalizável) mais facilmente legível; e uma diretoria *incoming*, com os dados que estão a ser recebidos vindos do *collector*. Um exemplo de cadeia de diretorias para encontrar dados no *presenter* de um certo *collector* é a seguinte: /*dsc/data/ServidoresPT/ns.dns.pt/2013-11-16/*.

Para a representação gráfica, o *presenter* usa o *ploticus*, uma biblioteca gráfica em C/C++. O trabalho da máquina *presenter* resume-se, então, a: receber dados vindos da comunicação remota com os *collectors*; extração da informação nos ficheiros XML para ficheiros *.dat*, lidos pela aplicação que os desenha (*ploticus*); construção dos gráficos consoante os *datasets* definidos, em função dos pedidos do utilizador que usa a página *Web* da aplicação. O *presenter* contempla ainda uma função de limpeza de registos antigos, para poupar espaço em disco.

A Figura 3, retirada do manual oficial da aplicação [7], representa esquematicamente o funcionamento do DSC:

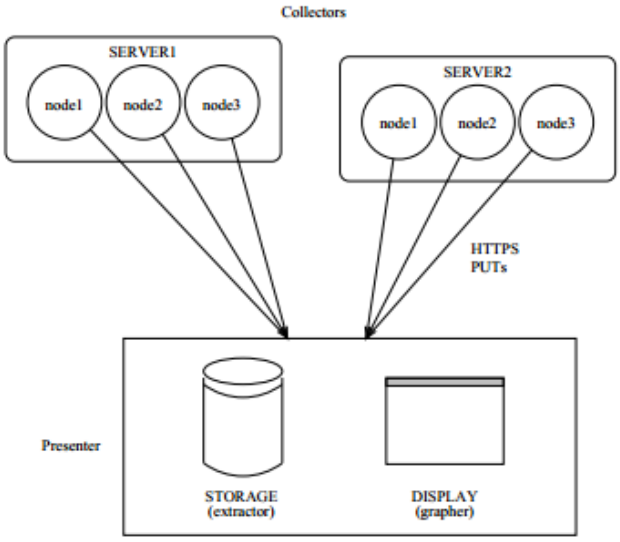
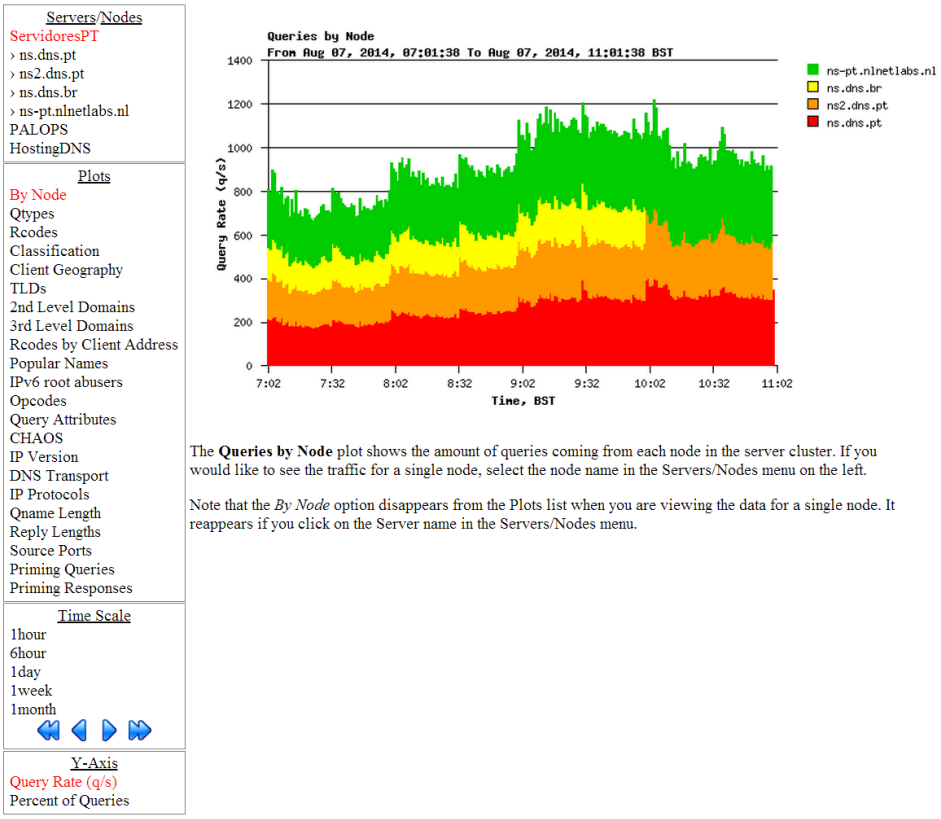


Figura 3 - Exemplo de funcionamento do DSC (collector e presenter)

Até à data de redação deste relatório, o DSC recolheu informação referente a mais de um ano de tráfego dos servidores de nomes da zona .pt. A Figura 4 mostra as estatísticas dos servidores de nomes portugueses numa semana de Agosto de 2014.



The **Queries by Node** plot shows the amount of queries coming from each node in the server cluster. If you would like to see the traffic for a single node, select the node name in the Servers/Nodes menu on the left.

Note that the *By Node* option disappears from the Plots list when you are viewing the data for a single node. It reappears if you click on the Server name in the Servers/Nodes menu.

Figura 4 - Dados estatísticos do DSC para os Servidores de Nomes Portugueses

Capítulo 3

Fase Final – Novo Sistema de Monitorização

3.1 Contextualização

Depois de uma fase preliminar (primeiros três meses do projeto) onde se incidiu principalmente na investigação e compreensão dos sistemas que constituem a infra-estrutura técnica da Associação DNS.PT, foi tempo de reestruturar, com base no conhecimento adquirido até então, a monitorização e alarmística dos sistemas, apresentando um novo sistema para o efeito, mais adaptado às necessidades e melhor preparado para o futuro.

Se numa fase inicial se incidiu mais em conhecer o protocolo DNS e como a Associação se envolve nele, nesta fase dar-se-á mais importância ao conjunto dos sistemas e serviços que a Associação tem, que servem de suporte direto ao DNS ou ao negócio, e de como integrá-los na monitorização e alarmística.

Neste capítulo explora-se todo o processo de desenvolvimento do novo sistema de monitorização, começando por avaliar o sistema antigo e planear o desenvolvimento do novo.

3.2 Levantamento de *Assets* (Sistemas e Serviços)

Este levantamento foi levado a cabo em duas partes: no início do projeto, como forma de perceber a infra-estrutura técnica, e aquando da necessidade de listar tudo aquilo que se queria monitorizar. Assim, criou-se uma listagem de máquinas e serviços que ilustra a composição da infra-estrutura técnica que se pretende monitorizar, de acordo com o que está presente na Tabela II da secção Anexo.

3.3 Levantamento de Requisitos

Para fazer uma avaliação competente e pertinente, foi fulcral saber o que se pretendia obter de um sistema de monitorização e alarmística, e só depois julgar o sistema existente. Para tal, foram feitos variados levantamentos de informações para conhecer todas as facetas que um sistema deste tipo apresenta.

Começou-se por fazer uma listagem compreensiva dos *Items* que existiam, dos sistemas e serviços que eram monitorizados e da estrutura do próprio sistema de monitorização. Quanto à lista de de *Items*, obtiveram-se, conforme apresentado em Anexo, na Tabela II, todos os que eram usados na monitorização e alarmística, ficando a conhecer-se o que poderia ser mudado e o que estaria apto para transitar para o novo sistema.

Para conhecer a estrutura da monitorização e compreender os seus processos, elaboram-se esquemas (presentes em Anexo nas Figuras I a V) que visam o conhecimento dos fluxos e técnicas de obtenção de informação de monitorização e alarmística.

Por fim, identificaram-se as formas existentes para obter informação dos *Hosts* configurados no sistema de monitorização. O protocolo SNMP e o próprio agente de comunicação *Zabbix* – ambos explicados no Capítulo 4, secção 4.1 – foram as duas principais formas de estabelecer canais de fluxo de informação dos *Hosts* monitorizados para o servidor de monitorização. Outro sistema que agradou muito devido à sua versatilidade foi o *Java* JMX. Esta tecnologia é aprofundada no Capítulo 4, secção 4.4.

Do levantamento de requisitos resultaram as seguintes conclusões:

Monitorização DNS – Capacidade de obtenção de:

- Número de *Queries* a chegar ao servidor de somes;
- Número de *Queries* a serem feitas pelo servidor de nomes;
- Número de *Requests* a chegar ao servidor de nomes;
- Tipos de *Queries* a chegarem ao servidor de nomes;
- Tipos de *Queries* a serem feitas pelo Servidor de nomes;
- Tipos de *Replies* a serem dadas pelo servidor de nomes;
- Quantidade de pedidos por protocolo (IPv4 / IPv6);
- Latência de reposta dos Servidores de Nomes da zona .pt

Monitorização SIGA – Capacidade de obtenção de:

- Estado das JVM (*Java Virtual Machine*), nos *Hosts* aplicáveis, quanto a:

- Espaço na *Heap* usado;

- Espaço na *Heap* livre;

- Estado da base de dados *ORACLE;*

- Estado da base de dados *MySQL;*

- Estado dos serviços *JBOSS* e *LIFERAY;*

- Estado dos *Web Services*

Monitorização dos *Hosts* – Capacidade de obtenção de:

- Estado do disco;

- Estado do CPU;

- Estado da(s) interface(s) de rede;

Este levantamento serviu, assim, para se julgar a situação do antigo sistema de monitorização e identificar que monitorização necessária estava em falta e que informação a ser obtida podia – ou não – ser migrada para o novo sistema de monitorização.

3.4 Avaliação do Sistema Existente

Tendo em conta as anteriores análises ao sistema existente, pretende-se agora fazer uma avaliação crítica.

3.4.1 Monitorização do sistema DNS

Para monitorizar o estado do serviço de nomes – *BIND* – usava-se o método descrito na Figura V em Anexo. A utilização do *rndc stats* permite recolher os dados de utilização do serviço *BIND* em cada servidor de nomes, agregando dados como número total e tipo de *Queries* recebidas, *Replies* dadas (total e parcial por tipo), e todo o conjunto de estatísticas que a própria ferramenta guarda. A forma atual para recolha dados era adequada, pelo que se decidiu manter no novo sistema de monitorização. Ficou, ainda, decido que se reveriam os *scripts* que operacionalizam a disponibilização da informação para integrar no sistema de monitorização.

3.4.2 Monitorização do sistema SIGA

A monitorização do sistema SIGA estava dividida em 4 partes:

- Monitorização da *JVM* das máquinas;

- Monitorização das bases de dados;

- Monitorização dos *Web Services*;

- Monitorização das máquinas.

Quanto à JVM, os dados obtidos vinham de um *Template*, pelo que seria necessário rever a aplicabilidade de alguns *Items* e verificar se o que era possível obter poderia ser integrado na nova solução. Após o levantamento feito, decidiu-se rever a monitorização por JMX, migrando apenas os *Items* que permitiam monitorizar o estado da JVM e que já vinham no *Template*, e definir-se uma nova abordagem usando *Custom MBeans*, conforme explicado no Capítulo 4, secção 4.3.

No caso das bases de dados, decidiu-se refazer todo o processo de obtenção de dados. Como o sistema usado – *ORABBIX* – ficou obsoleto com o lançamento do sistema *DBforBIX* (do mesmo criador do *ORABBIX*), decidiu-se usar o *DBforBIX*, não sendo o *ORABBIX* transitável para o novo sistema de monitorização.

Relativamente aos *Web Services*, que eram monitorizados via *Zabbix Web Items*, a sua utilização foi questionada. Embora a forma existente fosse usada eficientemente na monitorização dos *Web Services*, a descoberta das capacidades do JMX e dos *Custom MBeans* levantou algumas ideias para monitorizar os *Web Services*. Decidiu-se manter a monitorização dos *Web Services* da forma existente (via *Web Items*) e desenvolver uma forma adicional de monitorização através da interface JMX.

Por fim, quanto à utilização das máquinas que suportam o SIGA, verificou-se que a forma como era já feita a monitorização (usando o agente *Zabbix* para remotamente verificar o estados das máquinas) era apropriada, pelo que não se viu obstáculos em migrar integralmente a forma de obtenção para o novo sistema de monitorização.

3.4.3 Monitorização dos Hosts

Os *Hosts* já estavam num estado mais aceitável no que toca à situação do antigo sistema de monitorização. Por serem usados os *Templates* do *Zabbix* para obter os dados de monitorização de CPU, dos processos a correr na máquina, dos *checksums* dos ficheiros de configuração mais importantes (por exemplo, o */etc/passwd*), não aparentava haver problema em migrar integralmente. No entanto, descobriu-se que apenas os *Templates* por defeito não tinham capacidade de satisfazer as necessidades

requeridas (por exemplo, não haviam dados para estatísticas de utilização do disco, apenas o estado da ocupação do mesmo). Ficou então decidido que se integraria uma forma de obter estas estatísticas no novo sistema de monitorização.

Quanto às estatísticas de rede, estas eram obtidas através do agente *Zabbix*, que invoca na própria máquina o programa que apresenta os dados de utilização da interface de rede (comando *ps* da *Shell* de *Linux*). Por ser a forma mais útil e simples, decidiu-se manter e migrar para o novo sistema.

3.4.4 Veredito

Depois de avaliado o que havia a manter e o que seria implementado na fase de passagem do sistema antigo para o novo, chegou-se à seguinte conclusão:

Monitorização a manter: Toda a monitorização de *Hosts*, *Web Items*, Monitorização do serviço de DNS

Monitorização a rever: Monitorização da JVM, Monitorização das bases de dados

Monitorização a implementar: Novas formas de monitorizar *Web Services* com JMX

3.5 Desenho do Sistema

Depois do processo de levantamento de serviços, servidores e da monitorização associada, foi necessário pensar em como concretizar este novo sistema. Conceptualmente, o sistema poder-se-ia descrever da seguinte forma:

3.5.1 Servidores de Nomes (*BIND*)

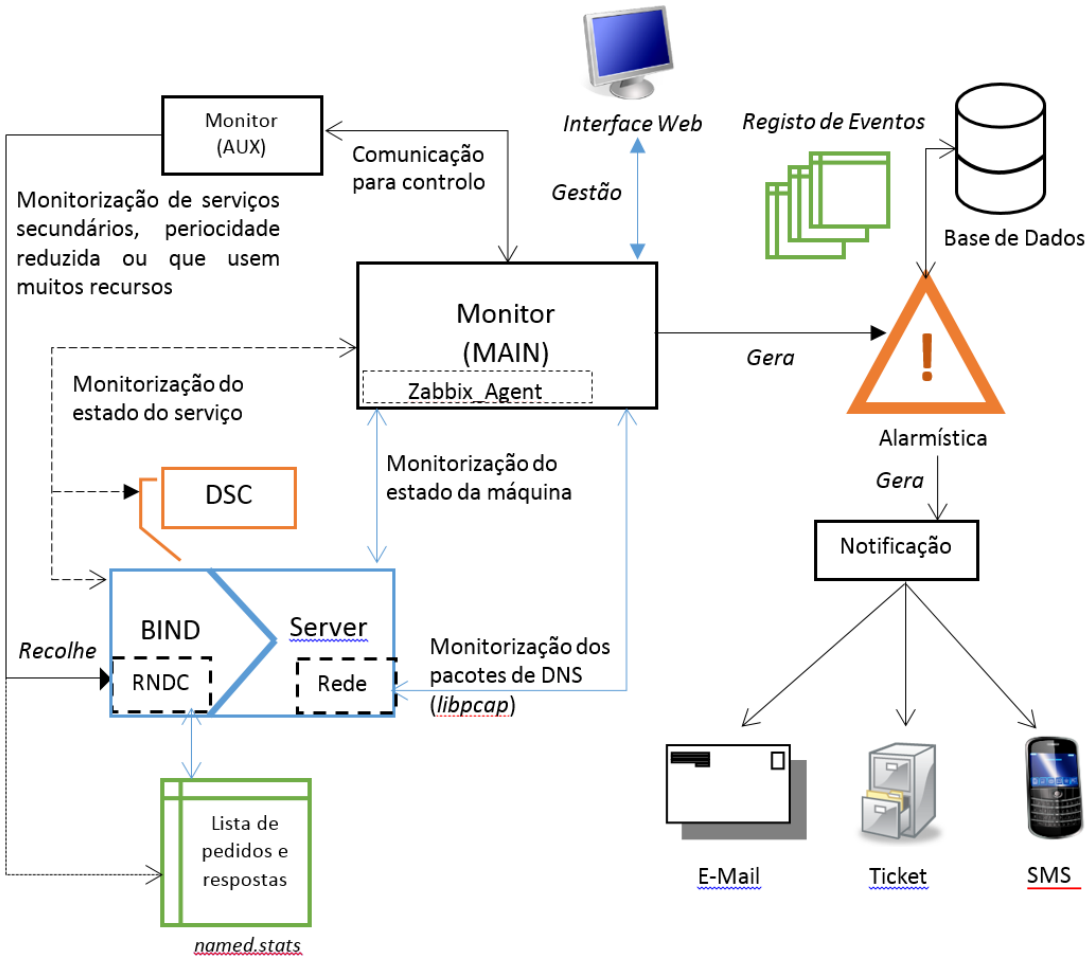


Figura 5 – Desenho do sistema de monitorização para Servidores de Nomes

Na Figura 5 identificam-se, facilmente, três grandes componentes: o monitor (que recolhe informação), o sistema monitorizado (que faculta informação) e a alarmística.

O monitor está dividido em monitor principal (*main*) e monitor secundário, pensando-se já na possibilidade de se descentralizar o sistema de monitorização. Com vários servidores de monitorização secundários que comunicam com um principal diminui-se a carga que um só monitor teria, mitigam-se eventuais falhas nos servidores e permite-se que a monitorização não falhe por completo se uma parte falhar.

Quanto ao sistema monitorizado, a recolha de dados ocorre a dois níveis: recolha dos dados de monitorização do sistema operativo e recolha dos dados de monitorização do serviço *BIND*. O sistema DSC está listado na Figura 5 mas não está associado ao sistema de monitorização como fonte de informação relativa ao *BIND*. É, no entanto, monitorizado o estado do serviço, pois por ser uma fonte externa de informação estatística, é importante saber se está a funcionar corretamente. O sistema operativo é

monitorizado, também, pelo monitor através do agente *Zabbix*. A monitorização da interface de rede, para além da realizada pelo agente *Zabbix* enquanto parte do sistema operativo, também é feita através da captura dos pacotes que chegam à interface e que dizem respeito ao serviço de DNS. Utilizando a biblioteca *libpcap* – que permite obter informações a nível dos pacotes TCP e UDP a chegar à interface de rede – pretende-se fazer uma monitorização adicional que não é feita nem pelo DSC nem pelo *rndc* (que é como se obtém as estatísticas do *BIND*).

Por fim, tem-se o sistema de alarmística, que é despoletado sempre que os dados de monitorização revelam padrões ou valores que evidenciam algum problema com um dado servidor ou serviço. A alarmística procede, então, à notificação da ocorrência, podendo ser via e-*mail*, SMS ou *ticket* (neste caso, o *ticket* será submetido no sistema de *trouble tickets* da Associação, o OTRS), e guarda registo desse evento numa base de dados (esta base de dados é a que o sistema *Zabbix* usa).

Toda a informação, bem como a gestão do sistema de monitorização, está presente na interface *Web* do *Zabbix*, facilitando a leitura e utilização do sistema de monitorização.

3.5.2 Sistema SIGA

O sistema SIGA assume um nível de complexidade superior ao dos servidores de nomes. O SIGA é composto por um conjunto de serviços distribuidos que necessitam de ser monitorizados heterogeneamente. Por exemplo, a monitorização dos *Web Services* requer uma tecnologia e abordagem diferente da monitorização do estado de um serviço ou da monitorização de uma JVM. Estas diferenças potenciam a utilização de diferentes técnicas e tecnologias para obtenção de informação, como a utilização da tecnologia JMX para obtenção informação da JVM ou mesmo para configuração de *MBeans*; ou a utilização de bibliotecas para comunicação com as bases de dados – quer seja *Oracle*, quer seja *MySQL* – como o *ORABIX ou DBforBIX*; ou mesmo apenas utilizando o agente *Zabbix* para recolher os dados de monitorização. Todas estas metodologias necessitam de ser configuradas e ajustadas para obter o melhor rendimento possível para a o sistema de monitorização.

Assim, definiram-se *a priori* algumas tecnologias a usar para monitorização dos diferentes serviços:

- Bases de dados: *ORABIX* ou equivalente (*DBforBIX*). Utilizar comunicação via JDBC (*Java Database Connectivity*, que é um *driver* que permite a ligação entre uma plataforma *Java* que envia *queries* SQL e uma base de dados relacional) para se ligar à base de dados *ORACLE* usada na atividade da Associação DNS.PT.

- Plataforma Java / JVM: Utilizar *Java Gateway* do *Zabbix* para se ligar à interface JMX nos servidores de conteúdos. A *Java Gateway* é mecanismo que a plataforma *Zabbix* tem para concretizar a ligação aos servidores JMX.

- Servidores de Conteúdos: Monitorizar ligações de rede e outras métricas do Sistema Operativo através das funcionalidades oferecidas pelo agente *Zabbix*.

A Figura 6 pretende ser um guia na estruturação da monitorização e alarmística para o sistema SIGA.

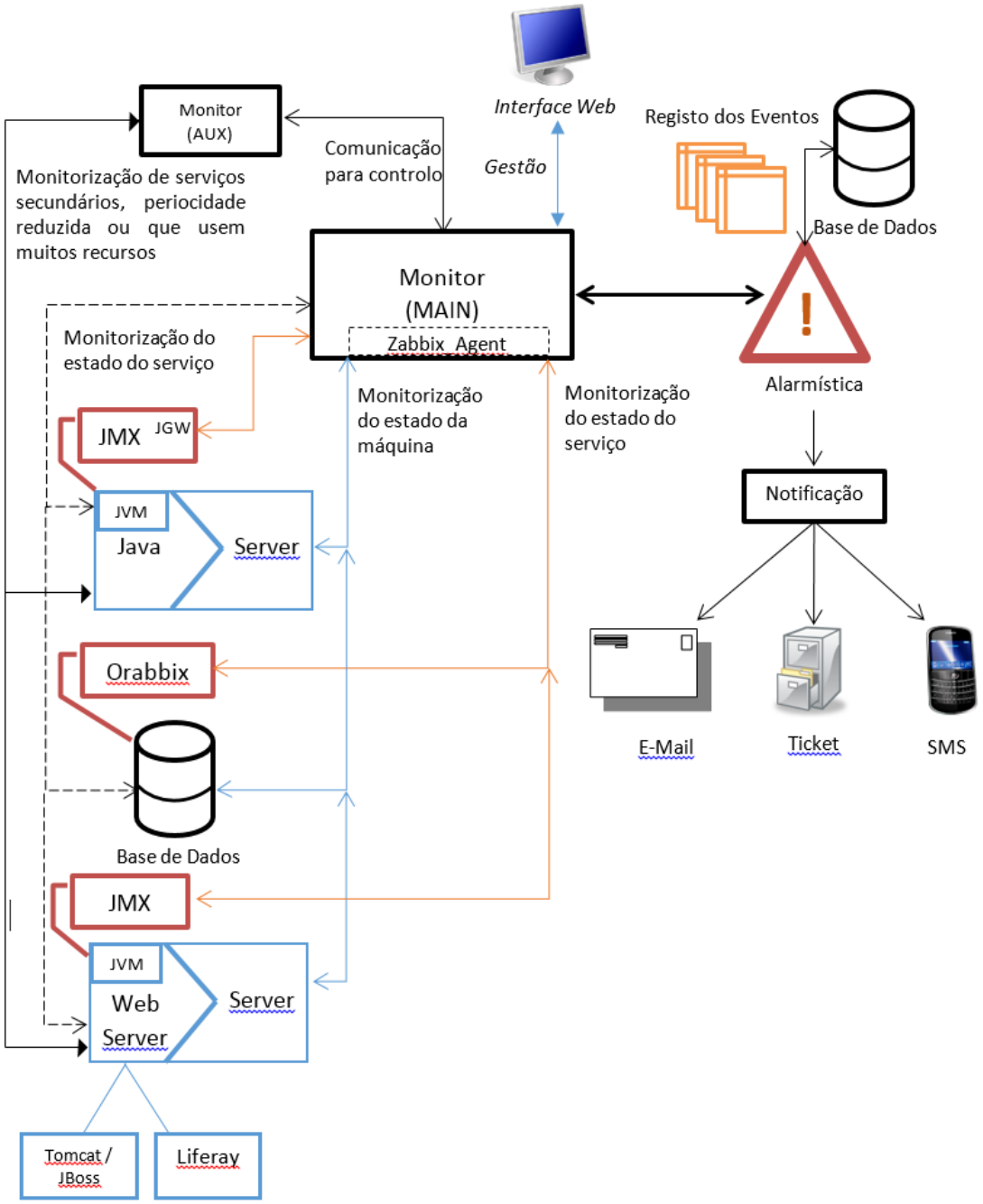


Figura 6 - Desenho do sistema de monitorização para sistema SIGA

Todos estes esboços dos sistemas representam uma abordagem inicial ao desenho do sistema final. Servem de guia para implementar uma lógica e, como tal, estão suscetíveis a alterações que se julguem necessárias.

3.6 Planeamento da Implementação

Para preparar todo o processo de implementação do novo sistema de monitorização foi necessário planear cuidadosamente várias etapas para dividir os vários processos que estariam envolvidos. Assim, definiram-se as seguintes etapas:

- Levantamento de requisitos para o servidor

- Instalação da máquina servidor

- Instalação do sistema de Monitorização *Zabbix*

- Configuração do sistema

Estas quatro etapas ditam o processo de instalação e configuração do servidor do sistema de monitorização. Passa-se agora a explicar o que se espera de cada etapa.

3.6.1 Levantamento de Requisitos para o Servidor

Nesta etapa adquire-se conhecimento de todas as tecnologias que são necessárias para a instalação do sistema de monitorização *Zabbix*. De acordo com os requisitos que podem ser encontrados na página *Web* do *software Zabbix*, o servidor necessita de:

- Sistema Operativo:

Linux,

IBM AIX,

FreeBSD,

NetBSD,

OpenBSD,

HP-UX,

Mac OS X,

Solaris,

Windows: 2000, Server 2003, *XP*, *Vista*, *Server* 2008, 7, 8, *Server* 2012 (*Zabbix agent* apenas).

As Tabelas 2 e 3 apresentam os requisitos de *software* para a instalação do servidor *Zabbix*.

- Base de dados do Servidor de Monitorização:

Software	**Versão**	**Considerações**
MySQL	5.0.3 ou superior	Se for usado para a base de dados do *backend*, é necessário *InnoDB engine*.
Oracle	10g ou superior	N/A.
PostgreSQL	8.1 ou superior	A versão sugerida para esta base de dados é, pelo menos, *PostgreSQL* 8.3, devido a performance.

Tabela 2 – Excerto da tabela de bases de dados recomendadas pelo *Zabbix*

- Extensões e Bibliotecas para o Servidor:

Requisito	**Considerações**
libssh2	Necessário para suporte a SSH. Versão 1.0 ou superior.
fping	Necessário para ICMP *ping Items*.
libcurl	Necessário para monitorização *Web*.
net-snmp	Necessário para suporte a SNMP.
OpenIPMI	Necessário para suporte a IPMI.
libiksemel	Necessário para integração com *Jabber*.

Tabela 3 – Tabela de requisitos de bibliotecas e extensões para o servidor

- *FrontEnd* (Servidor de Conteúdos):

    - *Apache* (Versão 1.3.12 ou Superior)

    - PHP (Versão 5.3.0 ou Superior)

- *Java*

- *Java SE Runtime Environment* (Versão 1.7.0 ou superior)

Destes requisitos levantados, optou-se pela seguinte configuração para o servidor:

- Sistema Operativo: *CentOS* 6.4. Este Sistema Operativo é já amplamente usado na Associação e é uma versão de Linux robusta e atual. Decidiu-se, portanto, que seria uma boa opção para hospedar o Servidor *Zabbix*.

- Base de dados: *MySQL*. Por ser um sistema de base de dados já utilizado no seio da Associação e por ter todas as funcionalidades que se desejam de uma base de dados, aliadas à forte documentação existente, optou-se por usar esta base de dados para o Servidor *Zabbix*. Outro fator que pesou a decisão foi o uso desta base de dados no sistema antigo de monitorização, e nunca terem existido problemas graves.

- Extensões e Bibliotecas: Da lista, apenas não se manteve *OpenIPMI* e *libiksemel* por não se ponderar o uso de IPMI (tecnologia de gestão de máquinas remotas usando ligação a este sistema em vez de se estabelecer ligação ao Sistema Operativo) e por não se querer utilizar o *Jabber* (serviço de *Instant Messaging* usado como forma de contacto para alarmística), respetivamente.

- *FrontEnd* e *Java*: Instalou-se tudo conforme os requisitos.

Foi também necessário decidir o espaço em disco disponível para a máquina. Este aspeto toma especial relevância se se tiver em conta que, sem um planeamento cuidadoso do tamanho inicial do disco, rapidamente se pode esgotar o espaço. Portanto, recorreu-se às fórmulas apresentadas na Tabela 4, para estimar o espaço em disco necessário.

Parâmetro	Fórmula para espaço em disco necessário (em *bytes*)
Zabbix configuration	Tamanho fixo (10 *Megabytes* aproximadamente).
History	**Fórmula**: dias*(itens/*refresh* rate)*24*3600**bytes* **itens**: nº de itens **dias**: nº de dias que os dados devem persistir. **refresh rate**: tempo médio de *refresh* dos itens recolhidos. **bytes**: nº de *bytes* para guardar um valor (dependendo da BD, cerca de 50 *bytes*).

Trends	**Fórmula**: dias*(itens/3600)*24*3600**bytes* **itens** : nº de itens **dias**: nº de dias que os dados devem persistir. **bytes**: nº de *bytes* para guardar um valor *trend* (dependendo da BD, cerca de 128 *bytes*).
Events	**Fórmula**: dias*eventos*24*3600**bytes* **eventos**: nº de eventos por segundo. Um (1) evento por segundo na pior das hipóteses. **dias**: nº de dias que os dados devem persistir. **bytes**: nº de bytes para guardar um valor *trend* (dependendo da BD, cerca de 130 *bytes*).

Tabela 4 - Tabela para estimação do espaço em disco necessário para o servidor [8]

De acordo com a tabela de referência (Tabela 4), estima-se que o espaço em disco seja o seguinte:

Dias: para a *History*, cerca de uma semana (7 dias) é necessário para se terem os valores exatamente como foram recolhidos. Para *Trends* e *Events* será suficiente guardar-se durante cerca de um ano (365 dias).

Itens: para calcular este valor, usou-se uma abordagem por comparação com aquilo que o sistema antigo tinha. Como no anterior sistema de monitorização existiam três servidores *Zabbix* (dois para monitorização específica e outro que condensava toda a monitorização), considerou-se que os valores que estes servidores recolhiam estavam obsoletos e/ou duplicados, pelo que se descartou o valor do servidor que apenas condensava informação. Concluiu-se, então, que cerca de 85% daquilo que existia na monitorização de Sistema Operativo e na monitorização das Aplicações era realmente útil, chegando ao seguinte valor estimado para o número de *Items*:

$$(604 + 2372) * 0.85 \approx 2529$$

Refresh Rate / Pedidos por Segundo: o *refresh rate* é o valor mais difícil de estimar, em muito devido ao tipo de monitorização heterogénea que se pretende implementar. Assim, como tanto existirão *Items* a serem recolhidos com uma periocidade inferior a um minuto, como existirão *Items* com periocidades superior a uma hora, decidiu-se usar os valores dos servidores de monitorização antigos para se ter uma abordagem mais realista. Assim, na mesma linha de raciocínio usado para o cálculo do número de *Items*, tem-se o seguinte valor de pedidos por segundo:

$$(7.38 + 22.68) * 0.85 \approx 26$$

De acordo com a fórmula, estima-se que se necessite do seguinte espaço para a *History*:

7 dias * (26 *Items*/seg) * 24 horas * 3600 segs * 50 *bytes* = 786240000 *bytes* ≈ 750 *Megabytes*

Para o espaço necessário para guardar os *Trends*, chegou-se ao seguinte valor:

365 dias * (2529 *Items* / 3600 segs) * 24 horas * 3600 segs * 128 *bytes* = 2835717120 *bytes* ≈ 2,6 *Gigabytes*

Já para os *Events*, estima-se pelo pior caso, usando o valor de 1 evento por segundo:

365 dias * 1 evento/seg * 24 dias * 3600 segs * 130 *bytes* = 4099680000 *bytes* ≈ 3,8 *Gigabytes*

Total = 10 *Megabytes* (Configuração) + 750 *Megabytes* (*History*) + 2,6 *Gigabytes* (*Trends*) + 3,8 *Gigabytes* (*Events*)

Total ≈ 7,14 *Gigabytes*

No total, espera-se usar cerca de 7,14 *Gigabytes* num ano de monitorização. Perante este valor, decidiu-se começar com 10 *Gigabytes* de espaço em disco para ter alguma margem de manobra para o futuro.

Posto tudo isto, o servidor *Zabbix* ficou com a seguinte especificação:

Nº de Processadores: 2

Memória RAM: 3 *Gigabytes*

Sistema Operativo: *CentOS* 6.4

Espaço em disco: 10 *Gigabytes*

Base de dados: *MySQL* 5.1

FrontEnd: *Apache* 2.2.15

PHP 5.3.3

BackEnd: *Java Runtime Environment* 1.7

libssh2, fping, libcurl, netsnmp

3.7 Instalação do Sistema

Depois de levantados todos os requisitos e decididas as especificações da máquina que irá hospedar o servidor *Zabbix*, passou-se à instalação.

A infra-estrutura técnica de servidores, na Associação DNS.PT, tem uma forte componente de virtualização. Existe um esforço para tentar, sempre que possível, criar servidores virtuais por todas as vantagens que isso traz.

Uma máquina virtual seria uma mais-valia pela facilidade de gestão (existe um sistema de gestão centralizada de virtualização, o *XenCenter* da *Citrix*), rapidez de criação de novos servidores (o *XenCenter* permite em poucos passos criar uma máquina apenas selecionando o Sistema Operativo e especificando os atributos técnicos da máquina, como número de CPUs, quantidade de memória e disco) e simplicidade de processos de *backup* (é possível tirar um *snapshot*, uma imagem do sistema operativo tal como está, para recuperar caso aconteça algo à máquina original).

Postas todas estas vantagens, criou-se uma máquina virtual com as características já apresentadas no levantamento de requisitos do servidor, que rapidamente ficou pronta para começar o processo de instalação e configuração do sistema de monitorização.

Instalação do *Software* necessário:

- Base de dados
- Apache
- PHP
- Java
- Instalação do *Zabbix Server*

A instalação decorreu nos seguintes passos:

1. Criação de um utilizador do Sistema Operativo *zabbix*, pertencente ao grupo de utilizadores *zabbix*.
2. Criação da base de dados – *MySQL* – para o *Zabbix Server*. Criação do *schema* da base de dados (descrição formal da estrutura da base de dados e descreve a sua organização e como está construída), criação das imagens usadas pelo sistema, e inserção dos dados iniciais.
3. Configuração das *sources* e instalação do servidor, com o seguinte comando para a configuração:

   ./configure --enable-server --enable-agent --with-mysql --enable-ipv6 --with-net-snmp --with-libcurl --with-libxml2

4. De seguida, passou-se à instalação do *Zabbix Server*, usando o comando

   make install

Com estes passos completos, ficou-se com um servidor de monitorização pronto a ser configurado de acordo com as especificações definidas, ficando a faltar a interface *Web*.

A interface *Web* do *Zabbix Server* necessita de um servidor de conteúdos (*Apache*) e de PHP para as funcionalidades das várias páginas. Assim, para instalar a interface, foi apenas necessário copiar os vários ficheiros que vinha no arquivo, para uma subdiretoria da diretoria padrão do *Apache*, a diretoria *zabbix*. Uma vez copiados os ficheiros, a página está pronta a visualizar assim que seja lançado o serviço *httpd*.

A restante configuração da interface *Web* é feita já no *browser*, onde aparece a listagem de pré-requisitos, conforme a Tabela 5, para a interface funcionar corretamente:

Pré-Requisito	Valor Mínimo	Descrição
Versão de *PHP*	5.3.0	
PHP memory_limit	128MB	No ficheiro de configuração php.ini: *memory_limit = 128Mb*
PHP post_max_size	16MB	No ficheiro de configuração php.ini: *post_max_size = 16Mb*
PHP upload_max_filesize	2MB	No ficheiro de configuração php.ini: *upload_max_filesize = 2Mb*
PHP max_execution_time	300 segundos	No ficheiro de configuração php.ini: *max_execution_time = 300*
PHP max_input_time	300 segundos	No ficheiro de configuração php.ini: *max_input_time = 300*
PHP session.auto_start	Tem de estar desabilitado	No ficheiro de configuração php.ini: *session.auto_start = 0*
base de dados	Uma das seguintes: IBM DB2, *MySQL*, *Oracle*, *PostgreSQL*, *SQLite*	Uma das seguintes bases de dados tem de estar instalada: ibm_db2, MySQL, oci8, pgsql, sqlite3

bcmath		php-bcmath
mbstring		php-mbstring
sockets		php-net-socket. Necessário para o support a *scripts*
gd	2.0 ou superior	php-gd. Extensão PHP GD tem de suportar imagens PNG (*--with-png-dir*), imagens JPEG (*--with-jpeg-dir*) e FreeType 2 (*--with-freetype-dir*).
libxml	2.6.15	php-xml ou php5-dom
xmlwriter		php-xmlwriter
xmlreader		php-xmlreader
ctype		php-ctype
session		php-session
gettext		php-gettext

Tabela 5 – Conjunto de requisitos para a interface *Web*

Depois de confirmar que todos estes pré-requisitos eram cumpridos, a interface apresenta a página como sugere a Figura 7.

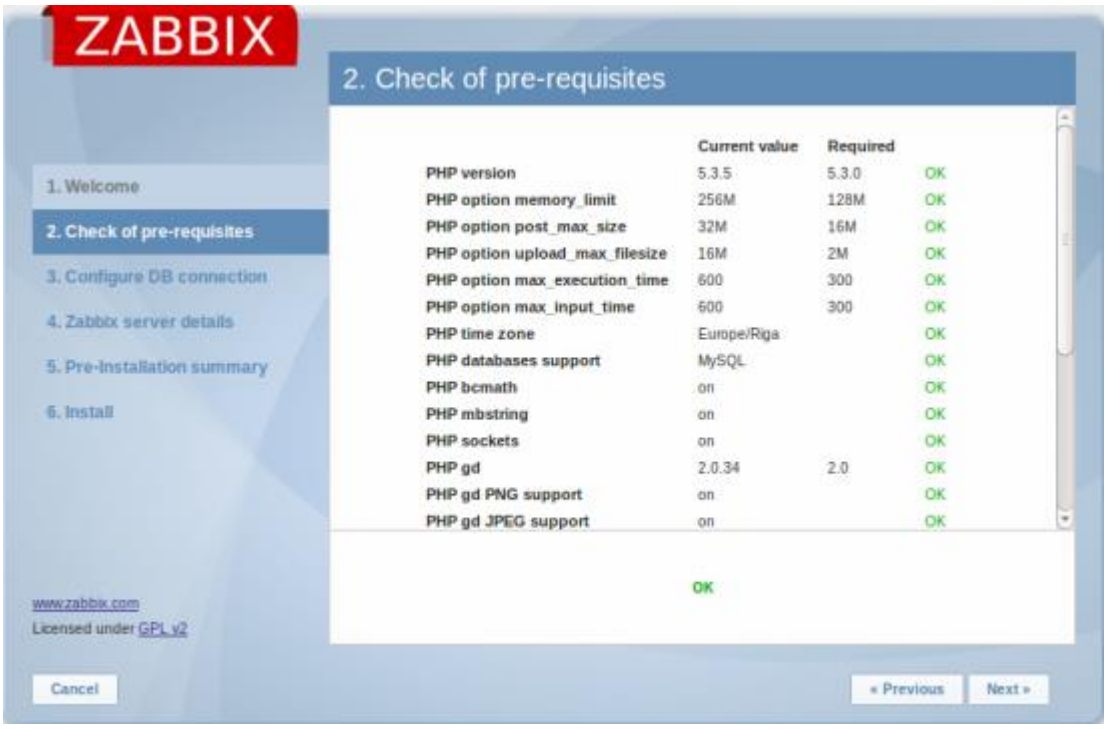


Figura 7 – Lista de pré-requisitos e estado destes.

Quando todos os itens estiverem a OK, pode-se seguir para a próxima fase, que se prende com as informações da base de dados a ser usada pelo servidor *Zabbix*.

No ecrã representado na Figura 8 introduzem-se os dados para configurar o acesso à base de dados *MySQL* de suporte ao servidor:

Figura 8 – Exemplo do ecrã para preenchimento dos dados da base de dados

Depois de testar a conexão e esta estar OK, passa-se à fase seguinte, que trata do preenchimento dos detalhes do servidor *Zabbix*.

A Figura 9 representa o ecrã de configuração do servidor, pedindo apenas três informações: Nome DNS do servidor, porta a usar, e nome a dar ao servidor (é relevante para identifica-lo no sistema de monitorização). Para o efeito, foram usadas as seguintes informações:

Host: <nome DNS do servidor>

Porto: 10051 (porto por defeito para o servidor *Zabbix*)

Nome: *Zabbix*Server

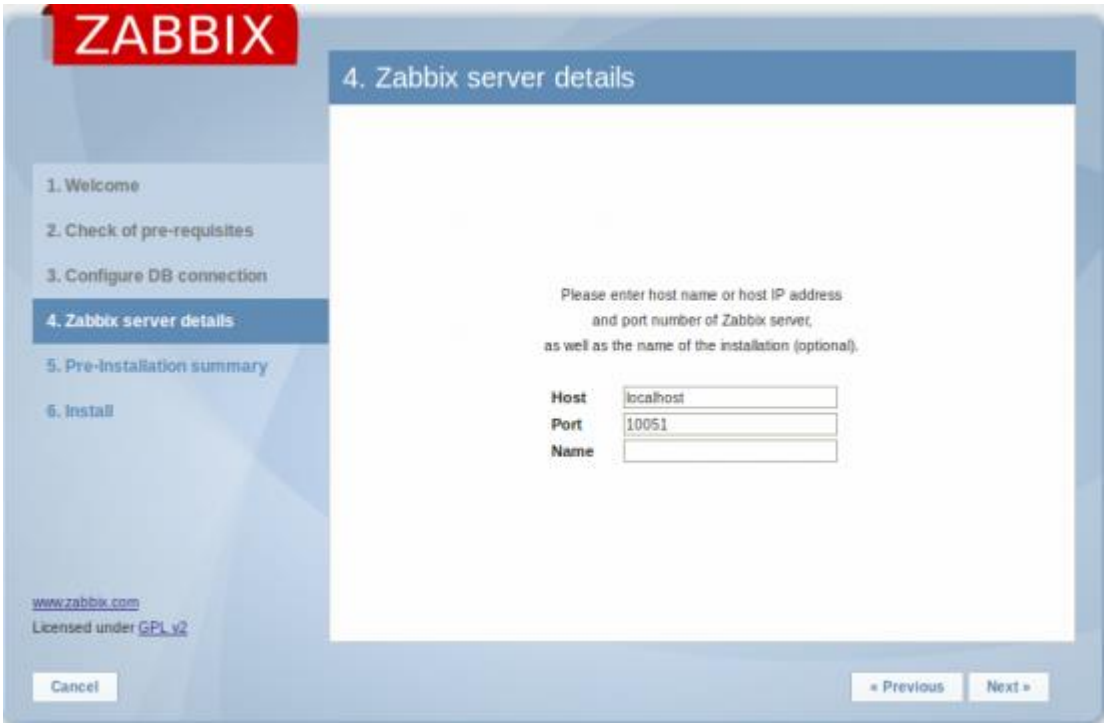


Figura 9 – Exemplo do ecrã para preenchimento dos dados do servidor

Uma vez fornecidas todas as informações solicitadas, pode-se passar ao próximo ecrã, que sumariza tudo o que foi feito nos passos anteriores. Revê-se se tudo está de acordo com o que se pretende para o sistema de monitorização e avança-se para a última etapa da configuração da interface *Web*.

A última etapa é relativa ao ficheiro de configurações PHP da interface gráfica. Neste ecrã apenas se testa se o ficheiro está OK ou se existe algum problema. É neste ecrã que se cria o ficheiro de configuração PHP, de nome *zabbix.php.conf*, fazendo *download* através da própria página. Uma vez guardado e confirmado que está tudo a postos para finalizar o processo (o estado passa para OK), aparece o botão *Finish*. Depois de clicar no otão *Finish*, o sistema fará a configuração da interface gráfica e o ecrã de *login* – presente na Figura 10 – deve aparecer se introduzirem as credenciais por defeito (utilizador *Admin*, palavra-passe *zabbix*).

Figura 10 – Ecrã de *Login* da interface *Web Zabbix*

Depois de todos estes procedimentos, o servidor de monitorização ficou totalmente operacional, com uma interface *Web* para gestão e consulta, e completamente apto a ser configurado para obtenção de métricas de monitorização, bem como a criação de alarmística associada as essas métricas recolhidas.

3.8 Configuração das Máquinas a Monitorizar

Para que uma máquina possa ser monitorizada no sistema *Zabbix*, ela tem de estar acessível pelo servidor. Por isso, a primeira preocupação foi garantir que o servidor de monitorização conseguia estabelecer comunicação no porto 10050 (porto padrão para o agente *Zabbix* estar à escuta) com a máquina a monitorizar. Para tal, foram abertas exceções na *firewall* para permitir estabelecer ligações vindas do servidor de monitorização para os agentes nas máquinas a monitorizar.

Normalmente, a comunicação estabelece-se do servidor para o agente, sendo o servidor que pede ao agente que lhe retorne os valores que pretende. No entanto, existe monitorização que requer que seja o agente a iniciar a comunicação com o servidor (*zabbix trapper*), pelo que existiu a necessidade de permitir que o servidor fosse contactado na porta 10051, vindo dos servidores monitorizados.

Depois de garantir o acesso às máquinas, começou-se a instalação dos agentes nas referidas máquinas. Para isso, configuraram-se as *sources* com o comando

./configure --enable-agent

seguido do habitual

make install

Esta operação instala o agente na máquina, com todos os binários, executáveis e ficheiros de configuração, e define ainda o agente como um serviço no Sistema Operativo, para que se possa, rapidamente, parar e reiniciá-lo. A configuração do agente é simples, sendo necessário fazer algumas mudanças no ficheiro de configuração [9] em relação ao que já vem por defeito. As alterações a fazer são:

EnableRemoteCommands: 1 em vez de 0

- Permite que se possam executar remotamente comandos na máquina. Especialmente útil caso se queira configurar algum *Item* que dependa do resultado de um comando. É preciso especial atenção ao dar esta habilidade ao agente pois fica-se vulnerável a executar comandos que podem, potencialmente, corromper a máquina em questão.

Server: *<IP do servidor de monitorização>*

- Necessário para saber de quem deve aceitar ligações e pedidos por informação.

Hostname: *<nome da máquina no sistema de monitorização>*

- Esta definição é crítica para o funcionamento do sistema. Quando o *Host* é criado no *Zabbix*, o nome tem de coincidir exatamente com o nome indicado no ficheiro de configuração, caso contrário o servidor não reconhece o *Host*.

Existe, ainda, mais uma configuração a fazer, talvez a mais importante em termos de utilidade para o sistema de monitorização, que é o *UserParameter* [10].

O *UserParameter*, que pode ser traduzido como Parâmetro de Utilizador, é um dos aspetos que torna o agente *Zabbix* tão versátil e completo, pois permite que se definam, de forma totalmente livre, *Items* que podem ser invocados remotamente e que retornam resultados dependendo da *script* que foi definida para tal. O seu raio de ação vai desde a simples invocação de um comando e retorno do valor resultante, até à execução de *scripts* que aceitam parâmetros e que podem até gerar alterações no próprio sistema. A capacidade do *UserParameter* é, para todos efeitos, limitada pelo próprio limite daquilo que se consegue fazer usando *scripts*, e pela quantidade máxima de dados que podem ser retornados numa execução, 512 KB [10].

Para configurar um *UserParameter* é necessário aceder ao agente na máquina em que se pretende recolher algum dado específico, e adicionar no ficheiro de configuração do agente – ficheiro *zabbix_agentd.conf* – a *keyword UserParameter* [10]. Para definir um novo *UserParameter* apenas se necessita de duas informações: a chave, que indica o

nome do *UserParameter* e que tem de ser única no conjunto de *UserParameters* para o agente em que se está a configurar, e o comando a ser invocado, que fornece o *output* do *UserParameter*. Seguem-se alguns exemplos de *UserParameter*, um sem argumentos e outro com argumentos:

UerParameter=exemple.custom.helloworld, echo "Hello World"

UerParameter=exemple.custom.echo[*], echo "$1"

O primeiro exemplo é um *UserParameter* simples, que apenas executa o comando *echo* para devolver uma *string*. Já o segundo exemplo é um pouco mais complexo, utilizando parâmetros para passar ao comando que se associa à chave do *UserParameter*. Neste caso, o comando *echo* vai usar o parâmetro passado como próprio parâmetro do comando. Como referido anteriormente, o comando que se associa a um *UserParameter* pode ser qualquer coisa, desde um comando básico da *Shell* de *UNIX* (neste caso), até a invocações de *scripts*, que encapsulam várias ações.

Esta funcionalidade é basilar para o sistema de monitorização e é amplamente usada na monitorização que recorre a meios mais avançados, como é o caso da monitorização do *BIND* (ver Capítulo 4, secção 4.2.2).

Capítulo 4

Concretização do Novo Sistema de Monitorização

Configurar o sistema de monitorização *Zabbix* é a parte mais importante e de maior complexidade deste projeto. Todo o processo de configuração requer a ponderação de vários fatores, sendo a alarmística e a forma de obtenção dos dados, os mais importantes a considerar.

4.1 Configuração dos *Hosts*

Uma das partes mais importante do sistema de monitorização são os *Hosts*. Todos os sistemas que produzam indicadores que se queira monitorizar têm de constar no sistema *Zabbix*. A Tabela II da secção Anexo foi o ponto de partida para a listagem dos *Hosts* a configurar.

Criar um *Host* é um processo simples, sendo apenas necessário utilizar as funções para tal na interface *Web* do *Zabbix*. Para configurar, é preciso saber três informações [4]: nome a dar ao *Host* no sistema de monitorização (tem de ser único no sistema de monitorização), endereço IP do *Host* (pode ser IPv4 ou IPv6) e o nome DNS do *Host*.

A documentação oficial do sistema *Zabbix* aconselha que seja usada, preferencialmente, a ligação por endereço IP, mas pode estar definido o endereço IP e o nome DNS. Estas duas informações são usadas na definição da interface de comunicação a usar pelo servidor, que podem ser as seguintes:

- Interface *Zabbix Agent* [4]: sendo a interface mais utilizada para monitorização, este modo de comunicação usa o agente que foi instalado nos servidores para recolher indicadores de várias formas, permitindo, por exemplo, acesso ao servidor para execução remota de comandos vindos do servidor de monitorização ou a definição de *UserParameters*.

- Interface SNMP [11]: O protocolo SNMP é o protocolo por excelência para efeitos de monitorização. Não é, no entanto, dada primazia à sua utilização devido às

variadas capacidades que o agente *Zabbix* apresenta e do facto de as máquinas utilizadas terem, na sua grande maioria, um agente *Zabbix* a correr. Ainda assim, este protocolo é vital para monitorização de *Hosts* que não possuam agentes *Zabbix*, como por exemplo *Routers* e outro equipamento que não tem sistema operativo compatível com o agente *Zabbix*.

- Interface JMX [12]: A interface JMX é utilizada para comunicação entre dois pontos usando tecnologia *Java*, que foi criada exatamente para gestão de variados recursos [12], desde a JVM até recursos locais na máquina ou mesmo conteúdos *Web*. É parte importante para monitorização de aspetos que o agente *Zabbix* não é capaz de saber. A sua utilização é referida ao longo deste documento, com ênfase durante a explicação da monitorização por *Java MBeans* (Capítulo 4, secção 4.3.1).

- Interface IPMI [13]: Esta interface permite o contacto com sistemas mesmo quando estes estão desativados. Através de uma ligação direta ao *hardware* (liga-se diretamente ao sistema em causa, sem ser necessário aceder pelo sistema operativo para contactar com ele, e independente do CPU da máquina), esta interface permite aceder às máquinas quando estas estão desligadas, possibilitando algo que raramente é possível para sistemas computacionais: aceder a eles mesmo quando estão desligados [13]. Não existem planos para monitorizar *Hosts* com esta tecnologia, pelo que a interface IPMI não tem utilidade, para já, para o sistema de monitorização.

Quando se cria um *Host* é necessário associá-lo a um grupo. Os grupos, neste novo sistema, agrupam *Hosts* que partilham o mesmo paradigma ou sistema, para que um grupo tenha *Hosts* que se relacionam entre si e com a informação associada ao nome do grupo. Por exemplo, o grupo *Linux* terá apenas *Hosts* cujo sistema operativo seja *Linux*, independentemente de serem servidores de nomes, servidores aplicacionais ou outros tipos de servidores. Esta divisão lógica dos servidores é importante para manter a organização e legibilidade do sistema.

No ecrã de criação de um *Host* (Figura 11), pode-se associar a grupos existentes ou criar um novo grupo, sendo que um grupo de *Hosts* (um *HostGroup* no sistema *Zabbix*) não tem grande quantidade de informação associada, apenas nome e, se se quiser, *Hosts* associados.

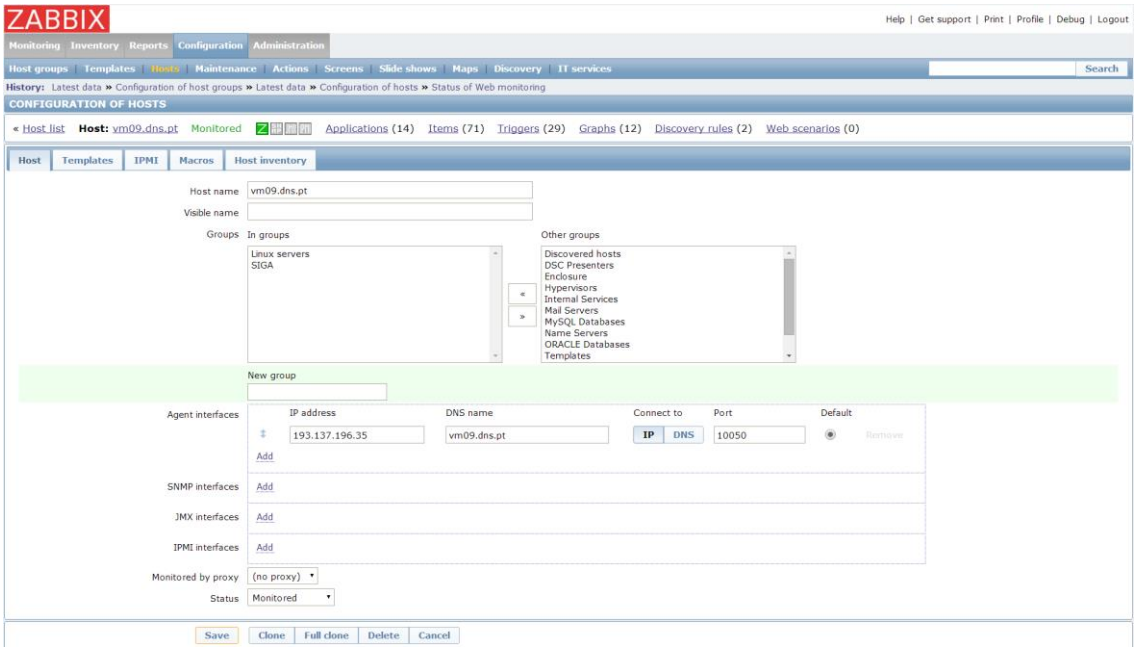


Figura 11 – Ecrã de criação de um *Host*

Esta configuração é das mais importantes no que diz respeito às bases do sistema de monitorização, sendo a outra, a definição dos *Items*. Uma vez configurados os *Hosts*, se tudo tiver corrido bem, o ecrã com a listagem de *Hosts* deverá aparecer conforme a Figura 12:

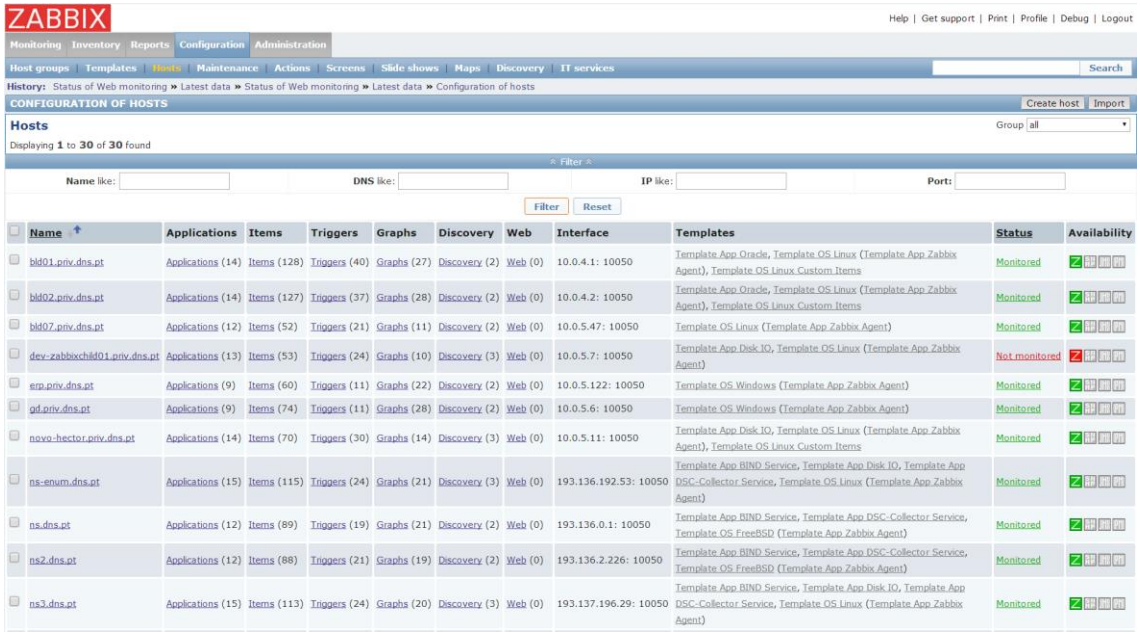


Figura 12 – Ecrã de *Hosts* com a listagem de alguns dos *Hosts* que integram o sistema

Depois de criados todos os *Hosts* necessários de acordo com o levantamento feito, passou-se à criação e configuração dos *Items*.

4.2 Configuração da monitorização e alarmística do *BIND*

A Tabela 6 serviu de guia no que toca aos vários dados que se necessitava recolher, como os vários *resource records* (vários tipos de informações que um domínio pode disponibilizar) úteis para saber o estado do *BIND*. Esta lista não é extensiva quanto a tipos de *resource records*, mencionando apenas os que são úteis para o efeito.

Incoming Queries

Hosts / Items	RESERVED0	A	NS	MD	CNAME	SOA	NULL	PTR	HINFO	MX	TOTAL
ns.dns.pt		●	●			●				●	●
ns2.dns.pt		●	●			●				●	●
ns3.dns.pt		●	●			●				●	●
ns4.dns.pt											
ns-enum.dns.pt								●			●
phobos.fccn.pt		●	●			●				●	●

Incoming Requests

Hosts / Items	QUERY	IQUERY	UPDATE	NOTIFY
ns.dns.pt	X	X	X	X
ns2.dns.pt	X	X	X	X
ns3.dns.pt	X	X	X	X
ns4.dns.pt				
ns-enum.dns.pt	X	X	X	X
phobos.fccn.pt	X	X	X	X

Legenda	
●	- Indicador a recolher
	- Indicador a não recolher
X	- A remover

Tabela 6 – Excerto das tabelas de *Items* a recolher para monitorização do *BIND*

Relembrando o que foi identificado no levantamento de requisitos da monitorização do *BIND*, temos as seguintes abordagens ao que foi proposto inicialmente:

- *Número de queries a chegar ao servidor de nomes*

  Alcançado com a obtenção dos totais de *resource records* a serem pedidos ao servidor de nomes.

- *Número de queries a serem feitas pelo servidor de nomes*

  Alcançado com a obtenção dos totais de *resource records* que são pedidos pelo servidor de nomes.

- *Número de requests a chegar ao servidor de nomes*

  Alcançado com a obtenção dos totais de *requests* que chegam ao servidor de nomes.

- *Tipos de queries a chegarem ao servidor de nomes*

  Alcançado com a obtenção dos valores, por segundo, de *queries* por dado *resource record*, ao servidor de nomes.

- *Tipos de queries a serem feitos pelo servidor de nomes*

  Alcançado com a obtenção dos valores, por segundo, de *queries* por dado *resource record*, que o servidor de nomes faz a outros servidores de nomes.

- *Tipos de replies a serem dadas às Queries feitas ao servidor de nomes*

  Alcançado com a obtenção dos totais de *replies* que o servidor de nomes está a enviar, como resposta às *queries* que lhe chegam.

- *Quantidade de pedidos por protocolo (IPv4 / IPv6)*

  Alcançado com a obtenção dos totais de pedidos em IPv4 e IPv6 que o servidor de nomes recebe.

Para concretizar o que se propôs anteriormente, foi necessário desenvolver uma forma de obter as informações citadas.

4.2.1 Configuração dos *Items*

O sistema antigo de monitorização obtinha os dados estatísticos do *BIND* pelo *rndc stats* [1] e pela filtragem desses dados, procurando apenas aquilo que se pretendia. De facto, as informações que esta ferramenta providencia vão muito além de apenas os tipos de *queries* feitas ao servidor de nomes (parciais e totais por tipo), pelo que se achou importante que fosse integrado no sistema *Zabbix* toda a informação útil que fosse possível obter.

Assim, resolveu-se criar um novo *script* para obter o que se queria da ferramenta *rndc stats*, de nome *parser.sh*, com o seguinte funcionamento:

1) Executar o *rndc stats*
2) Filtrar os últimos dados estatísticos (por defeito, a ferramenta acrescenta ao fim do ficheiro *named.stats* as estatísticas cumulativas para a última execução)
3) Percorrer o ficheiro e retirar para ficheiros distintos os vários grupos de informação presentes
   a. Criação dos ficheiros

      CacheDBRRsets;
      IncomingQueries;
      IncomingRequests;
      NameServerStatistics;

OutgoingQueries;
ResolverStatistics;
SocketIOStatistics;
ZoneMaintenanceStatistics.

b. Preenchimento dos ficheiros com os dados referentes a cada grupo anteriormente referido.

4) Remoção dos ficheiros temporários de apoio

As capacidades que este *script* tem, face ao que existia anteriormente, recaem principalmente na facilidade de obter os diversos dados para monitorização de cada categoria. Agora, para saber quantas *queries* chegam a um certo servidor de nomes, basta aceder ao ficheiro *IncomingQueries* e procurar pelo *resource record* pretendido. Usando comandos de pesquisa *UNIX* como o *grep*, pode-se rapidamente saber quantas *queries* pelo *resource record* SOA foram feitas no total, por exemplo. Similarmente, se se quiser saber quantos pedidos chegaram a um certo servidor de nomes em IPv4, basta aceder ao ficheiro *NameServerStatistics* e procurar por "IPv4 *requests received*".

Com esta funcionalidade abre-se um vasto leque de opções sobre aquilo que a dado momento é possível saber para o funcionamento do *BIND*. Para se ter o valor que se pretende, porém, tem de ser configurado um *UserParameter* específico em cada servidor de nomes, que permite o *Zabbix* obter remotamente a informação pretendida. Configuraram-se um conjunto de *Item keys* para identificar os recursos a obter, de acordo com o seguinte:

- *bind.incoming.query[*]*: Esta chave, que aceita como argumento a abreviatura do *resource record* pretendido, permite ao servidor de monitorização obter o valor total de *queries* por esse *resource record* que chegaram ao servidor de nomes.

- *bind.outgoing.query[*]*: Análogo ao anterior, devolve o total de *queries* por *resource record* que o servidor de nomes efetuou.

- *bind.incoming.queryTotal*: Esta chave define o *UserParameter* responsável por devolver o total de todas as *queries* que chegaram ao servidor de nomes.

- *bind.outgoing.queryTotal*: Igual ao anterior, mas devolve o total de *queries* que o servidor de nomes efetuou.

- *bind.incoming.requestTotal*: Devolve o total de *requests* que chegam ao servidor de nomes. Um *request* é identificado pelo *opcode*, podendo ser, entre outros, uma *query*, um *update*, um *notify*, etc.

- *bind.reply.nxdomain/nxrrset/successful/authoritative/nonAuthoritative*: Esta *key* define um conjunto de *UserParameters* para obter os tipos de resposta que o servidor de nomes está a devolver relativamente aos pedidos que estão a chegar.

- *bind.request.ipv4/ipv6*: Este indicador permite obter o total de pedidos feitos ao servidor em IPv4 e IPv6, respetivamente.

Com esta implementação, faltava apenas configurar na interface *Web* do *Zabbix* como recolher estas informações.

No sistema de monitorização criou-se o seguinte *standard*: para definir a monitorização de um sistema ou serviço, primeiro cria-se um *Template* [4] para o efeito, onde se definem os indicadores a recolher (através das chaves que identificam o dado a obter), a alarmística a associar e ainda os gráficos com a representação dos dados obtidos. Desta forma, a configuração da monitorização e alarmística fica abstraída da concretização, podendo-se adicionar e remover *Hosts* facilmente, pois uma vez aplicado um *Template* a um *Host*, este herda todos os *Items*, *Triggers* e *Graphs* que o *Template* define.

Configuração na interface *Web*

O primeiro passo é, então, criar o *Template*, de nome *Template App BIND service* (Figura 13). O nome escolhido segue uma convenção que se definiu para os nomes dos *Templates*, que deve começar com *Template*, seguindo-se o tipo do alvo monitorizado, que neste caso é uma aplicação. Depois disto segue-se o nome do serviço, *BIND*.

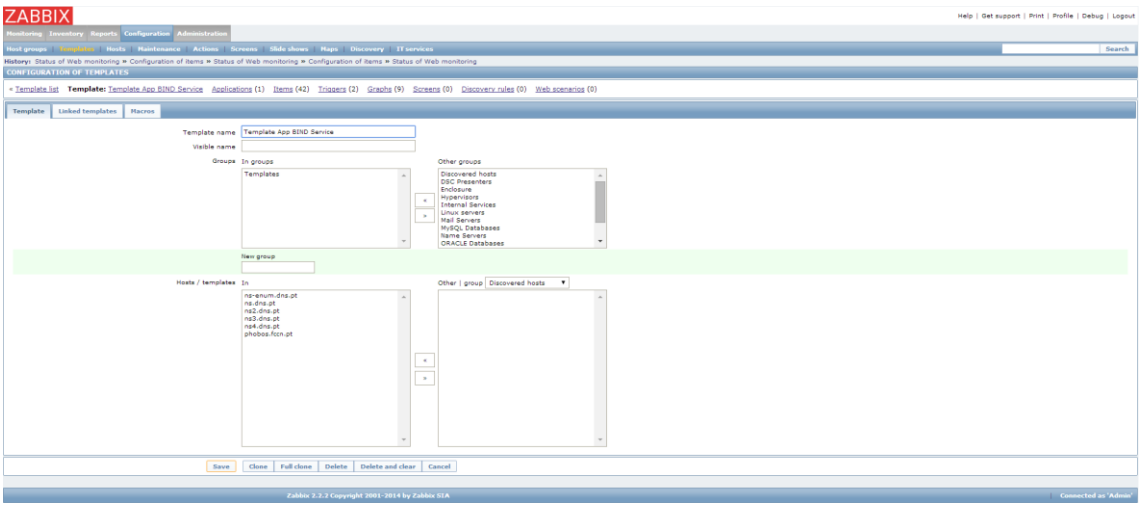


Figura 13 – Ecrã de criação de um *Template*

No conjunto de *Hosts* a associar, indicam-se a que *Hosts* este *Template* será aplicado. Como já se encontravam configurados, associaram-se logo os servidores de nomes. O passo seguinte é configurar os *Items*, usando as *Items Keys* criadas para o *BIND*.

Saber o total, até ao momento, de *queries* feitas ao servidor de nomes não é muito interessante, mas saber quantas estão a ser feitas por segundo, por exemplo, tem muito mais utilidade. Para isso, tem de se fazer o cálculo delta de dois valores consecutivos e depois dividir esse valor pelo tempo de atualização dos dados em segundos. Por exemplo, se se obtiverem os dados ao minuto, o cálculo delta a dividir por 60 segundos dará a quantidade de dados por segundo, que é o que se pretende. Felizmente, o servidor *Zabbix* faz essa operação sem necessidade de ajuda externa, ao disponibilizar um pós-processamento dos dados em duas formas: *delta (speed per second)* e *delta (simple change)*. O último apresenta o valor como diferença de dois valores consecutivos.

O intervalo com que os dados são actualizados é outra questão importante. Neste contexto, o ideal seria ter informação atualizada acerca destes *Items* o mais próximo possível de tempo-real. Como esta opção não é viável – a quantidade de dados que seria enviada e a capacidade de processamento necessária para lidar com isso não traria vantagens – optou-se por pôr um intervalo relativamente baixo, 30 segundos. Ter duas medições por minuto permite ter uma boa base para estimar o número de *queries* por segundo que se podem esperar chegar a certo servidor de nomes.

As Figuras 14 e 15 mostram como se processa a criação de um *Item* na interface *Zabbix*.

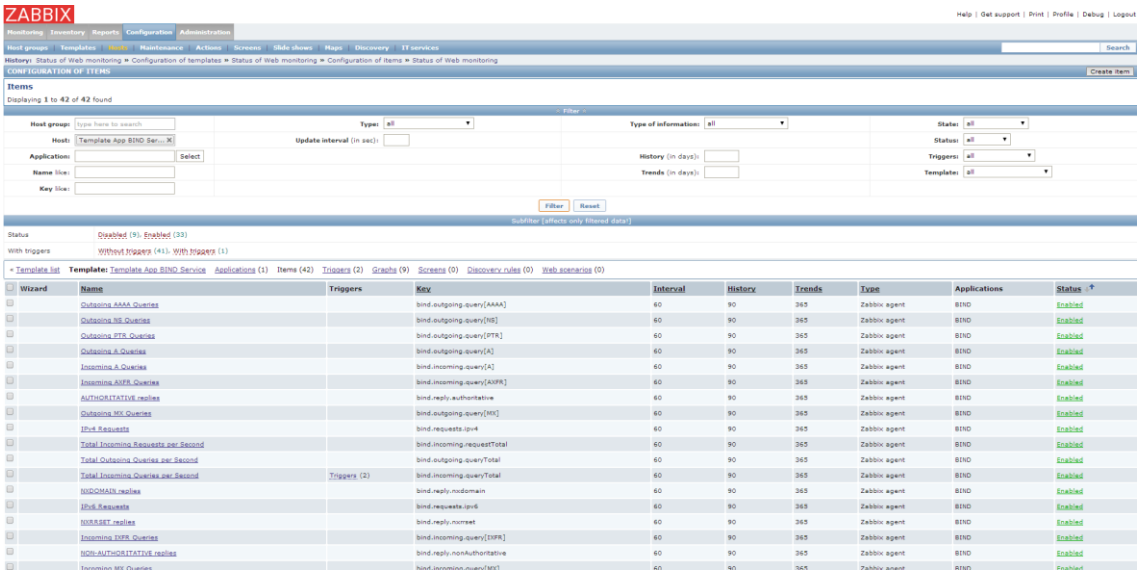


Figura 14 – Ecrã de criação de *Items*

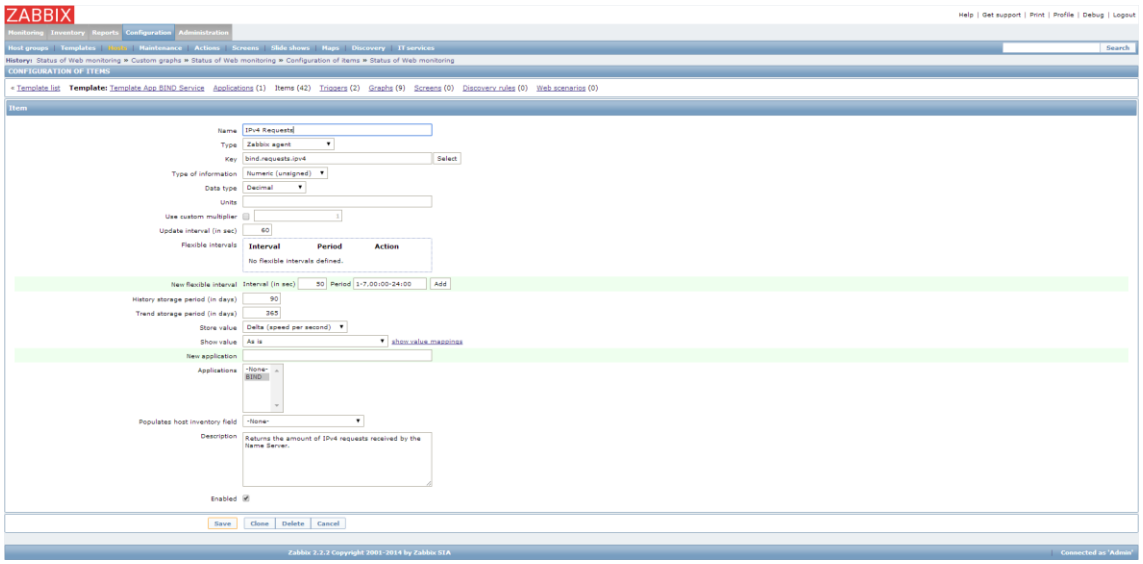


Figura 15 – Ecrã de configuração de um *Item*

Findo o processo de configuração dos *Items*, passou à configuração de *Triggers*, que faz parte da alarmística do sistema.

4.2.2 Configuração dos *Triggers*

Os *Triggers* têm dependência dos *Items*, pois necessitam dos dados que estes fornecem para avaliar condições de alarme [3]. A definição de *Triggers* passa sempre por definir uma função que, dependendo do valor que o *Item*, despolete um alarme ou não. São sempre de cariz booleano e existe um número limitado, mas grande, de condições pré-definidas a que um *Item* pode ser submetido.

Para este caso, a função mais usada foi a que verifica o valor dos últimos dados recebidos e os compara para verificar se são superiores a certo limite imposto ou se a diferença denota um aumento muito grande. Os *Triggers* definidos para o *BIND* têm como objetivo verificar aumentos bruscos (através da media diária) e aumentos muito significativos (através do limite fixo de um valor) nas *queries* que se recebem. O *Trigger* para a média é ainda limitado por um valor fronteira para evitar que em períodos de baixa utilização haja ocorrências desnecessárias.

Todos os *Triggers* têm uma severidade associada para que possa existir diferenciação naquilo que a alarmística reporta. Podem-se ter *Triggers* de cariz informativo, sem que estes apresentem um problema. As outras severidades permitem definir o grau do problema, para que se possa distinguir e agir de acordo com o problema que é apresentado.

As Figuras 16 e 17 mostram como se configuram *Triggers* na interface *Zabbix*.

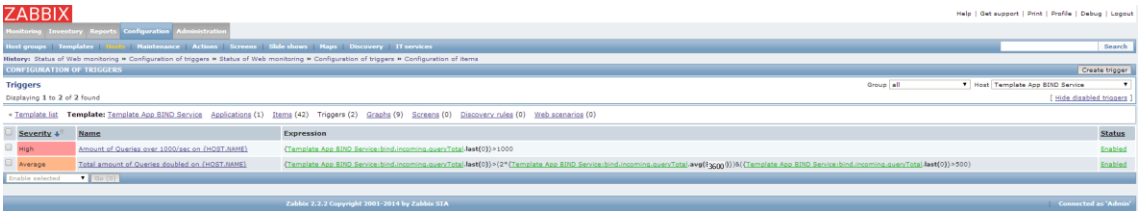


Figura 16 – Ecrã de criação de *Triggers*

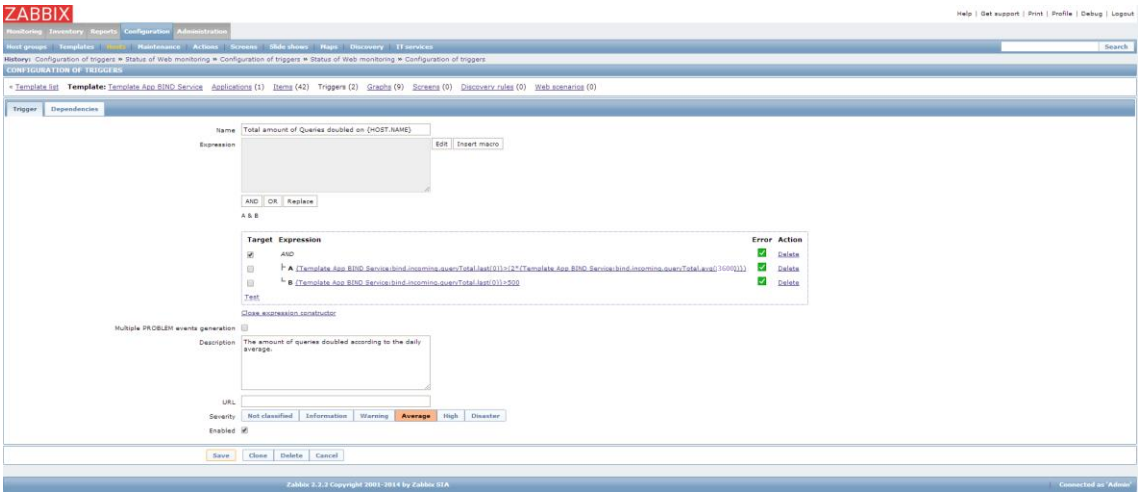


Figura 17 – Ecrã de criação das condições para despoletar alarmes

4.2.3 Configuração dos *Graphs*

A outra vertente de monitorização definida nos *Templates* é a questão da representação da informação. A forma predileta é o uso de gráficos com os valores que o sistema de monitorização recolhe, existindo uma forte componente gráfica no ecrã gigante usado na Associação DNS.PT.

A configuração de *Graphs* requer que já existam *Items* definidos. A criação de um *Graph* é simples, sendo apenas necessário decidir que *Items* incluir no gráfico e que tipo de gráfico se pretende criar (também se pode alterar as cores que cada *Item* toma na representação), deixando o resto das opções de configuração com os valores padrão [4]. Uma das opções que é útil por vezes mudar é o tipo de gráfico, podendo escolher entre normal (gráfico de linhas que unem os pontos referentes aos valores obtidos para dado *Item*), normal empilhado (a origem para cada *Item* não é a origem do referencial mas o valor do último ponto recolhido, ou seja, desenha-se sempre para cima em relação ao último *Item*, o que leva a que se tenha um gráfico empilhado, onde o maior valor no

eixo dos YY corresponde à soma de todos os valores para o valor no eixo dos XX) e cilíndrico (em inglês, *pie chart*, representa informação dividida em "fatias" de um todo).

Configuraram-se os *Graphs* (Figuras 18 e 19) de acordo com os *Items* recolhidos, agregando num único gráfico a representação de, por exemplo, todos os tipos de *query* que chegam a um servidor de nomes. Como os *Graphs* estão definidos num *Template*, todos os *Hosts* os vão herdar, tendo assim um conjunto de *Graphs* relacionados com o *BIND* para cada servidor de nomes.

Figura 18 – Ecrã dos *Graphs*

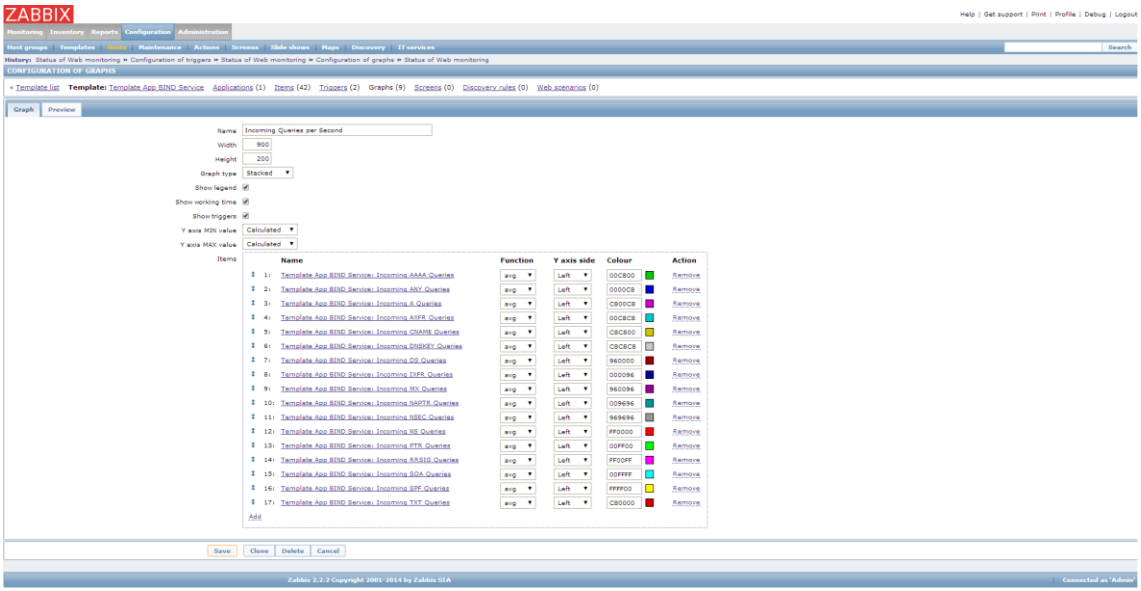


Figura 19 – Ecrã de criação de um *Graph*

No último exemplo, na Figura 19, pode-se ver o conjunto de *Items* que compõem este *Graph* e as cores que cada um assume, podendo-se editar as cores de cada um e escolher se se pretende apresentar o mínimo, a média ou máximo que cada *Item* possa ter para cada medição (apenas é aplicável para *Items* que tenham mais de uma medição para cada instante).

Este conjunto de *Items*, aliados aos *Triggers* definidos, constituem a monitorização e alarmística para os servidores de nomes, quanto à aplicação *BIND*.

4.3 Configuração da monitorização e alarmística do SIGA

O sistema SIGA é mais complexo e heterogéneo em termos de monitorização do que a aplicação *BIND*, devido ao conjunto de sistemas que o constituem, e da necessidade de várias metodologias de monitorização para a recolha dos indicadores.

Do levantamento levado a cabo, obtiveram-se as seguintes necessidades no que toca à monitorização do sistema SIGA:

- Estado das JVM (*Java Virtual Machine*), nos *Hosts* aplicáveis, quanto a:
    - Espaço na *Heap* usado

        Usando as capacidades da tecnologia JMX, obter o espaço utilizado na *Heap* da JVM.
    - Espaço na *Heap* livre

        Usando as capacidades da tecnologia JMX, obter o espaço livre na *Heap* da JVM.
- Estado da base de dados *ORACLE*

    Recorrendo a ligações à base de dados (via *Driver* de ligação – ODBC), obter estatísticas de performance do motor de base de dados e do estado das tabelas e acessos.
- Estado da base de dados *MySQL*

    De forma análoga à base de dados *Oracle*, obter informações sobre o estado da base de dados quanto ao motor desta e ao estado interno (estrutura).
- Estado dos serviços *JBOSS* e *LIFERAY*

    Monitorizar o estado dos serviços através da monitorização do estado dos portos de comunicação com as aplicações e observação dos processos.
- Estado dos *Web Services*

    Monitorizar via *MBeans* ou através da funcionalidade de *Web Monitoring* (*Web Items*) do *Zabbix*, auxiliado pelo sistema *Jenkins*.

4.3.1 Monitorização da JVM

Começando com a monitorização da JVM, definiram-se alguns indicadores a recolher, baseado-se no que já existia no antigo sistema de monitorização, no que foi investigado acerca desta tecnologia e no do que útil se poderia obter para a monitorização do estado da JVM.

A monitorização via JMX – interface para monitorização da JVM – necessita, primeiro, que o servidor onde corre a JVM tenha a opção de JMX ativada e esteja habilitado a ser acedido para recolher os dados. Para tal, aquando da execução da aplicação *Java*, deve-se adicionar as seguintes opções, destinadas à JVM:

-Dcom.sun.managment.jmxremote.port=9999: Indica o porto remoto para comunicação JMX.

-Dcom.sun.managment.jmxremote.ssl=false: Indica que não se usa a camada protocolar SSL para proteger a ligação.

-Dcom.sun.managment.jmxremote.authenticate=false: Indica que não se usam credenciais para autenticar os acessos à interface JMX.

Estes três parâmetros, quando passados no comando para executar a aplicação *Java*, expõem a interface JMX para se poder recolher os dados para monitorização da JVM. De notar que se prescindiu das camadas de segurança por dois motivos: o primeiro, e mais importante, é por se tratar de comunicação entre máquinas na rede interna que já possuem proteção da *firewall* local e da *firewall* corporativa. O segundo motivo é por questões de simplicidade, dado que acrescentar proteção à ligação adiciona complexidade que se achou, na altura, não compensar, visto ser algo que ainda era experimental.

Depois de se garantir o acesso aos dados para monitorização da JVM, passou-se à parte mais importante, que é recolher os indicadores. O levantamento feito anteriormente resultou numa listagem mais completa com tudo aquilo que se pretendia saber para a JVM de cada servidor integrante do SIGA, conforme a Tabela 7 demonstra.

	Java (JVM)					
Hosts/Items	# Loaded Classes	Heap Memory Used	PermGen Used *	Process CPU load	JVM Uptime	Thread Count
pro-lb01						
pro-bo01	●	●	●		●	●
pro-bo02	●	●	●		●	●
pro-bo03	●	●	●		●	●
pro-bo04	●	●	●		●	●
pro-fe01	●	●	●		●	●

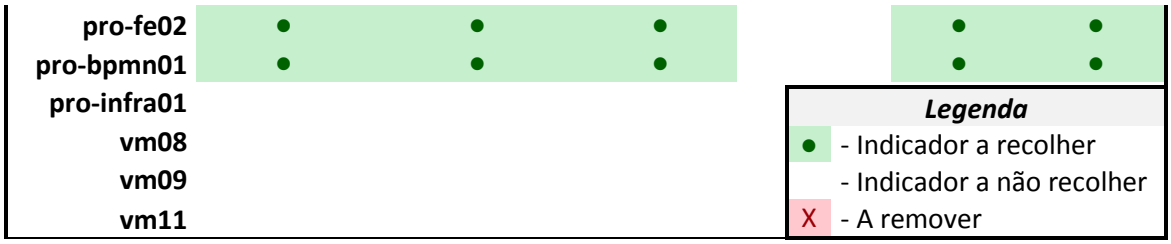


Tabela 7 – Excerto do levantamento da monitorização da JVM

De notar que um dos indicadores que se pretendia recolher era a carga de CPU gerada pelo processo da JVM. Este indicador foi, posteriormente, removido das intenções de monitorização por não ser, por um lado, algo que faça sentido monitorizar do ponto de vista da JVM, mas sim pelo Sistema Operativo, e por outro lado, não se saber até que ponto é que esse indicador seria apropriado para a monitorização da JVM (ficava sempre a incógnita se a carga era relativa ao panorama geral do CPU, se era apenas do CPU reservado para a JVM, sendo algo que não se conseguiu perceber).

Os indicadores recolhidos para o estado da JVM são todos importantes para ter a noção, a qualquer momento, de como está o estado da JVM. À parte da carga de CPU para o processo, que já foi abordada no parágrafo anterior, os outros indicadores prendem-se com o desenrolar das ações que o programa *Java* executa e são próprios do ciclo de vida da aplicação.

O número de classes carregadas (*# Loaded Classes*) e o número de *Threads* (*Thread Count*) são importantes para monitorizar os recursos que a JVM está a utilizar. Um aumento ou diminuição nestes valores pode indicar um comportamento anormal do programa ou da própria JVM, sendo que a contagem das *Threads* usadas pode revelar que o programa está a necessitar de mais ou menos recursos que o habitual. Já quanto à memória da própria JVM (*Heap Space*) [14], a sua monitorização é indispensável para se saber quando, por exemplo, a JVM ficou sem mais espaço para alocar a informação que vai sendo criada durante a execução do programa.

Quanto ao *PermGen* [14], parte da memória da JVM guardada para informação que subsiste às sucessivas execuções do *Garbage Collector*[2] da JVM, é um indicador importante a recolher devido à importância que tem para as aplicações com tempos de execução muito prolongados. Uma monitorização ativa do valor de *PermGen* permite perceber se é necessário aumentar o espaço dedicado na JVM, o que melhora a *performance* do programa (este indicador foi recolhido no seguimento de alguns

[2] *Garbage Collector* é a funcionalidade do *Java* que permite libertar o espaço em memória que já não é utilizado pelo programa ou sequer pela JVM.

episódios onde problemas na aplicação *Java* foram resolvidos com o aumento do espaço reservado para o *PermGen*).

Para recolher os dados via JMX utilizou-se o *Template* genérico que a aplicação *Zabbix* já disponibiliza e que contém aquilo o necessário para satisfazer os requisitos de monitorização da JVM.

Para obter todos estes indicadores, é necessário que as seguintes premissas se verifiquem:

1) A JVM a monitorizar tem a interface JMX capaz de ser contactada;
2) Existe um *Host* no sistema de monitorização com uma interface JMX definida;
3) O servidor de monitorização tem uma *Java Gateway* em execução.

A primeira premissa foi satisfeita com a passagem dos três parâmetros à JVM. A segunda premissa foi satisfeita aquando da criação dos *Hosts*. Quanto à terceira premissa, foi necessário executar a *Zabbix Java Gateway*, que vem no conjunto de *sources* do sistema *Zabbix*. Esta aplicação é responsável por tratar dos pedidos por indicadores JMX num *Host*, fazendo ela os pedidos usando a API de gestão de JMX fornecida pela *Oracle*. Para instalar e configurar a *Zabbix Java Gateway* necessário fazer o seguinte:

- Da mesma forma como se instalou o servidor e o agente *Zabbix*, aceder aos *sources* e configurar a instalação da *Java Gateway*

  ./configure –enable-java –prefix=<install_dir>

- Depois é só preciso instalar, correndo o comando

  make install

- Para iniciar a *Java Gateway*, basta correr o *script* criado na instalação

  ./startup.sh

Depois de ter a *Java Gateway* funcional, é necessário configurar o servidor *Zabbix* para que este saiba como a contactar. Para isso, é preciso editar o ficheiro de configuração do servidor (*zabbix_server.conf*) e alterar os seguintes campos:

JavaGateway=<*IP do Servidor*>

JavaGatewayPort=10052

Uma vez feitas as alterações, é necessário reiniciar o servidor *Zabbix* (para que as alterações sejam assimiladas) e o sistema de monitorização está apto a recolher os dados via JMX. Se tudo tiver corrido bem, deverá aparecer uma indicação visual do sucesso na comunicação via JMX, de acordo com as Figuras 20 e 21.

Figura 20 – *Hosts* com interface JMX configurada e acessível corretamente

Figura 21 – Ícone (JMX) que ilustra o sucesso na comunicação via JMX para certo *Host*

4.3.2 Monitorização da base de dados *Oracle* e *MySQL*

A configuração da monitorização das bases de dados do sistema SIGA pode ser condensada numa só secção, devido à forma encontrada de obtenção dos indicadores para cada base de dados. Embora completamente diferentes e com necessidades de monitorização distintas, a forma encontrada para as monitorizar foi apenas uma – *DbforBIX* [15].

Começando pelos requisitos levantados para a monitorização, as necessidades de conhecimento do seu estado são relativamente simples: saber se a base de dados está saudável. No caso das bases de dados *Oracle*, isto é conseguido através da monitorização dos seguintes indicadores:

- *Hit Ratio* nas tabelas (quantidade de pedidos que são feitos e recaem sobre informação já em *cache*);

- Número de sessões ativas;

- Valor de *PGA* (*Process/Program Global Area* é uma zona em RAM onde se guardam dados e informações de controlo para um processo, sendo monitorizado o espaço usado dessa zona);

- Tentativas falhadas de acesso à base de dados.

Este conjunto de indicadores é suficiente para se estar a par de eventuais problemas, sendo que a alarmística recai nestes indicadores (por exemplo, quando há uma falha na autenticação aquando do acesso à base de dados, a alarmística é despoletada, criando-se um evento com as informações relativas ao *login* falhado).

Já no caso das bases de dados *MySQL*, a monitorização recai nos *Items* definidos por defeito pela aplicação *DBforBIX*, uma vez que a base de dados *MySQL* não tem o mesmo nível de importância que as bases de dados *Oracle*. A alarmística que vem por

defeito com os *Templates* da aplicação *DBforBIX* para esta base de dados também foi a usada.

É importante salientar que a monitorização das bases de dados, quer *Oracle*, quer *MySQL*, ainda não foi revista e analisada cuidadosamente. Isto deve-se ao facto de pretender que esta análise seja feita por um administrador de bases de dados, que tem os conhecimentos e competências para discernir o que deve e não deve ser monitorizado, e só depois configurar a monitorização e alarmística de acordo com o seu *feedback*.

Monitorização Existente (antiga)			
ORACLE	*Manter?*		*Manter?*
Alive	✓	Max Sessions	✓
Archivelog	✓	Miss Latch	✓
Audit	✓	OrabbixVersion	X
CPU Util %	X	PGA	✓
CPU Util % Classic Idle	✓	PGA Aggregate target	✓
CPU Util % Classic System	✓	PHI/O Datafile Reads	✓
DB Block Gets	✓	PHI/O Datafile Writes	✓
DB Consistent Gets	✓	PHI/O Redo Writes	✓
DB Files Size	✓	Pin hit ratio - BODY	✓
DB Hit Ratio	✓	Pin hit ratio - SQLAREA	✓
DB Physical Reads	✓	Pin hit ratio – TABLE-PROCEDURE	✓
DB Size	✓	Pin hit ratio - TRIGGER	✓
DB Version	✓	Pool dict cache	✓
FRA Size	✓	Pool free mem	✓
FRA Used	✓	Pool lib cache	✓
Free memory	✓	Pool misc	✓
Hit ratio - BODY	✓	Pool sql area	✓
Hit ratio - SQLAREA	✓	Processes	✓
Hit ratio - TABLE/PROCEDURE	✓	Processor load	✓
Hit ratio - TRIGGER	✓	Processor load15	X
LI/O Block Change	✓	Processor load5	X
LI/O Consinstent Read	✓	Session Active	✓
LI/O Current Read	✓	Session Inactive	✓
Locks	✓	Session System	✓

Tabela 8 – Excerto do levantamento da antiga monitorização *Oracle*

De acordo com a Tabela 8, fez-se um levantamento para a base de dados *Oracle* quanto aos indicadores existentes, tendo sido feita posteriormente uma apreciação do que se devia e não devia manter. No entanto, a forma como se obtinham os indicadores no sistema antigo (via *ORABBIX*, que foi descontinuado em detrimento do *DBforBIX*) ficou obsoleta, verificando-se na instalação do *DBforBIX* que existiam *Templates*

bastante completos para a monitorização pretendida. Assim, decidiu-se usar o *Template* até receber o administrador de bases de dados.

4.3.3 Monitorização dos serviços *JBOSS* e *LIFERAY*

A monitorização dos serviços *JBOSS* e *LIFERAY* é feita de uma forma simples, recolhendo os indicadores apresentados na Tabela 9:

Hosts/Items	Port 80 Status	Port 443 Status	Port 700 Status	Port 3121 Status	Apache Service Status	Tomcat Service Status	JBOSS Service Status	Liferay Service Status
pro-lb01	✓	✓			✓			
pro-bo01								✓
pro-bo02							✓	
pro-bo03							✓	
pro-bo04							✓	
pro-fe01								✓
pro-fe02								✓
pro-bpm01								
pro-infra01								
vm08					✓	✓		
vm09			✓	✓	✓			
vm11						✓		

Tabela 9 – Levantamento dos indicadores a recolher para monitorização

A monitorização destes serviços é feita em dois aspetos: se é possível aceder a esses serviços e se os processos, relativos a estes serviços, estão efetivamente a correr. Para esta monitorização utilizou-se, em grande parte, os recursos que o próprio *Zabbix* já disponibiliza para monitorização de portos e processos, sem recorrer a *scripts* personalizadas, como em casos anteriores. Para a verificação do estado dos processos (por exemplo *Apache*, mas aplica-se também a *Tomcat*, *JBOSS* e *Liferay*) pode-se fazer de uma de duas formas:

a) Saber quantos processos existem que contenham certo padrão no nome

Ex.: *system.run[ps auwx |grep httpd | grep -v "grep httpd" | wc -l]*

b) Saber quantos processos com estado *running* existem para certo *user*

Ex.: proc.num[<name>,<user>,<state>,<cmdline>], onde

name – nome de processo (por defeito é "all processes")

user – nome de utilizador (por defeito é "all users")

state – *all* (por defeito), *run*, *sleep*, *zomb*

cmdline – filtrar por comando (através de expressão regular)

A versão "a)" fornece uma visão global dos processos que estão a correr com o nome ou *owner* "*httpd*" (serviço *Apache*), enquanto a versão "b)" permite distinguir o estado do processo com o nome ou *owner* pretendido. Acabou-se por implementar as duas formas, usando a forma a) para monitorizar o número de processos *Apache* e a forma b) para verificar os estados dos processos (*run* e *sleep*). Desta forma, tem-se uma visão geral dos processos existentes no *Host* para cada serviço e uma visão mais detalhada para cada um desses processos. A Figura 22 exemplifica a monitorização do *Apache*.

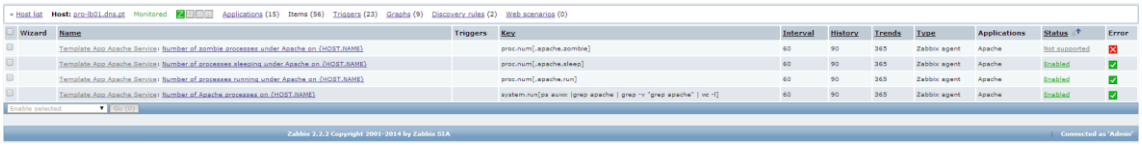


Figura 22 – Ecrã de *Items* para monitorização do serviço *Apache*

A monitorização dos portos de comunicação via *Zabbix* consiste na verificação se o porto está em *LISTEN state* e se é possível conectar-se. A obtenção dos indicadores é feita das seguintes formas:

net.tcp.listen[700] – Este indicador permite saber se o porto 700 está à escuta, ou seja, está em *LISTEN mode*.

net.tcp.port[<*default*>,700] – Este indicador permite saber se é possível estabelecer ligação ao porto 700.

Com estes dois indicadores é possível saber a qualquer momento se, por exemplo, se o porto 700 (porto destinado à comunicação do protocolo EPP, usado na comunicação entre *registries* e *registrars*) está apto a ser utilizado na comunicação. O porto 3121 é usado para testes de EPP, daí se monitorizar também da mesma forma como se monitoriza o porto 700. Os portos 80 e 443, por serem de protocolos bem conhecidos na utilização da *Web*, têm indicadores específicos que já vêm nos *Templates* do *Zabbix* para monitorização do serviço HTTP e HTTPS. As chaves dos *Items* são

net.tcp.service[http] e *net.tcp.service[https]*

e oferecem as mesmas informações que os outros indicadores referidos oferecem.

4.3.4 Monitorização dos *Web Services* SIGA

A monitorização dos *Web Services* é um aspeto importante para o conhecimento do estado do sistema SIGA, sendo necesário saber a qualquer altura, se os serviços estão a funcionar corretamente.

A primeira abordagem à monitorização foi usando a funcionalidade de *Web Items* do sistema *Zabbix*. Esta forma de recolha de indicadores para conteúdos *Web* já era usada na versão anterior do sistema *Zabbix*, sendo apenas necessário avaliar o que já existia quanto à relevância para se poder integrar no novo sistema. Fez-se um levantamento de tudo o que era recolhido, resultando na elaboração da Tabela 10.

	Jobs	STATE
Jobs Jenkins 2	CreateUsers	●
	ElectronicInvoices	●
	GenerateCSV	●
	MonitoringAlerting	●
	MonitoringAlerting - Renew	●
	PaymentERP	●
	RedunicreProcess	●
	SMSSender	●
	ProofDocsGenerator	●
	EvaluateTechnicalInfo	
	LetterSender	
	NotificationTransform	
	ReportsGenerator - Auditoria Financeira	
	ReportsGenerator - Dispersão Geográfica	

Legenda	
●	- Recolher
	- Não recolher
	- Enabled
	- Disabled

Tabela 10 – Excerto da lista de *Web Items* existentes no antigo sistema de monitorização

Do levantamento realizado apenas alguns *Items* foram mantidos. Para chegar a esta conclusão foi necessário julgar a validade destes indicadores, reunindo com outros elementos da equipa técnica para decidir quais eram importantes ter e quais não apresentavam relevância para o conhecimento do estado do sistema.

Uma vez decidido o conjunto de *Web Items* a configurar no novo sistema de monitorização, passou-se à implementação deles, cuja criação e configuração difere bastante em relação aos restantes *Items*.

Um *Web Item* é um conjunto de indicadores para um cenário de navegação numa página, que descrevem o sucesso ou insucesso de cada etapa desse cenário. Para configurar, é necessário descrever que página irá ser utilizada (indicando o URL), qual o código HTTP e resposta da página esperadas, para além de definir a periocidade para efetuar estas verificações.

As Figuras 23 e 24 mostram como configurar *Web Items* na interface gráfica do *Zabbix*.

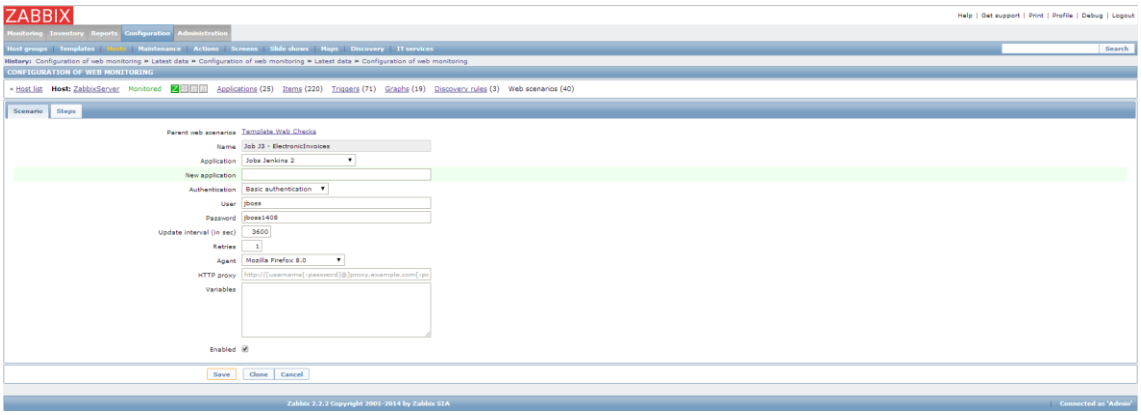


Figura 23 – Ecrã de criação de um *Web Item*

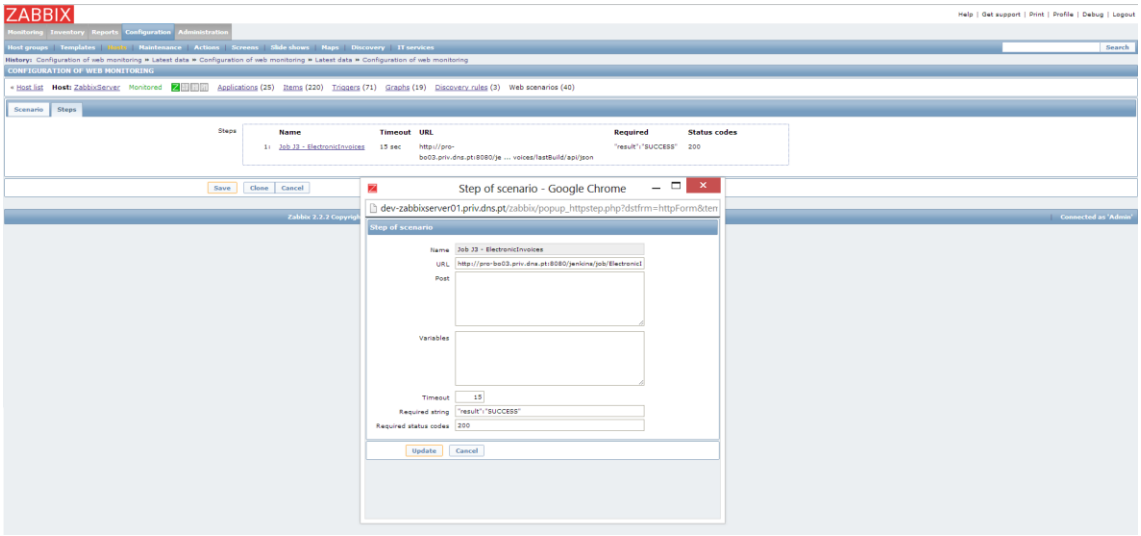


Figura 24 – Ecrã de especificação dos passos a executar no cenário *Web*

Os *Web Items* para monitorização dos *Web Services* do sistema SIGA fazem-se valer do sistema *Jenkins* para verificar a execução de tarefas que o sistema faz periodicamente. Estes *Jobs* – como são apelidados pelo *Jenkins* – produzem um *output* que é depois usado na monitorização do sucesso ou insucesso das tarefas.

4.4 Monitorização de outros Serviços *Web* da Associação

Existem na Associação DNS.PT outros serviços *Web* para além daqueles que o SIGA disponibiliza. De seguida, cobrem-se os casos que não foram abrangidos pela monitorização via *Web Items*.

4.4.1 *Java MBeans* e o Sistema de Monitorização

Por já existir conhecimento em programação *Java* e alguns servidores em questão já terem *Java* instalado, achou-se útil utilizar a tecnologia *MBeans* [16] do *Java* – disponibilizada em todas as JVM – e integrar os dados que se obtêm diretamente no sistema de monitorização *Zabbix*.

Um *MBean* (*Managed Bean*) é um objeto *Java* que representa um recurso que pode ser gerido, como uma aplicação ou um serviço. Este recurso expõe uma interface de gestão que consiste num conjunto de atributos (de leitura e / ou escrita), operações que podem ser invocadas e uma descrição própria. Estas características advêm dos padrões de *design* definidos pela especificação JMX [12], pretendendo criar um *standard* no que toca a especificações para desenho de aplicações de gestão de recursos [16].

Para concretizar um *MBean*, são necessários os seguintes componentes:

- Uma interface que seja conforme com a especificação JMX;

- Uma classe que implemente a interface descrita;

- Uma classe servidor, que operacionalize a classe definida e a disponibilize para acesso.

Segue-se uma breve explicação de cada uma das componentes.

Interface: *nomeMBean*

A especificação impõe que o nome da interface apresente explicitamente a intenção de usar *MBeans* [16], daí o *nomeMBean*.

A interface declara dois tipos de funções: relativa a atributos e operações genéricas. É através desta especificação que se pode inferir quais recaiem sobre atributos (pelos prefixos *get* e *set* dos métodos) e quais as operações (geralmente sem nenhum prefixo ou sufixo associado). As operações podem ou não alterar os valores dos atributos, dado que a especificação não é rígida quanto ao que estas operações podem fazer.

Classe: *nome*

A classe que implementa os métodos declarados pela interface tem como nome o nome da interface sem o sufixo *MBeans*. Assim, para a interface *XyzMBeans*, a classe que a implementa chamar-se-ia *Xyz*.

Esta classe é responsável por implementar os métodos declarados, respeitando as operações e atributos conforme a especificação dita. Se a interface tiver dois métodos, *getA* e *getB*, então a classe terá de apresentar dois atributos, A e B. A especificação de *getters* indica que estes métodos apenas devolvem os valores destes atributos, pelo que não deverão ter quaisquer argumentos aquando da invocação do método nem realizar nenhuma ação que possa alterá-los. No que toca a *setters*, estes deverão alterar o valor dos atributos a que se referem tendo, normalmente, um argumento, que é o novo valor a dar ao atributo em questão ou desencadear uma ação que o altere.

Agente JMX: *Server*

O agente JMX é o responsável por gerir os recursos que foram registados (na documentação da *Oracle* [16] apelida-se como instrumentados) pelo *MBeans*. A principal componente deste agente JMX é o servidor *MBean*, que gere os tais recursos. A classe que concretiza o servidor não está suscetível a especificações, pelo que pode ter qualquer nome.

O ponto mais importante deste servidor é o registo dos *MBeans* criados. Para isso, é preciso criar o servidor de *MBeans* onde se vão registar as operações, sendo para tal necessário obter uma instância de servidor via *ManagementFactory*. Esta *factory* consiste num conjunto de métodos estáticos (independentes das instâncias de objetos) que retornam plataformas *MXBeans* (*MBeans* que apenas permitem tipos que estejam em *java.lang.**), que representam a interface de gestão de uma componente da JVM.

4.4.2 Monitorização com *MBeans*: Motivação e Balanço

Este sistema apenas será relevante se existir forma de utilizar estes *MBeans* proactivamente num ambiente de monitorização. Assim, as interfaces de *MBeans* devem ser construídas com o intuito de obter informação para alimentar o sistema de monitorização, principalmente em aspetos que o atual sistema de monitorização é incapaz ou deficiente no desempenho dessa função.

Os *MBeans* cobrem algumas das lacunas na monitorização por agentes ou por SNMP. Fortes candidatos para este tipo de monitorização são:

- Sistemas Operativos: Informação não-habitual, como *registry keys* (*Windows*), informação acerca de ficheiros ou processos, dados de performance de componentes que podem não apresentar integração simples com sistemas de monitorização.

- Serviços: Saber informações do desempenho dos serviços em execução ou saber se podem ser executados. Através do PID de um processo, por exemplo, há muitas possibilidades no que toca a recolher informação. Pode-se interagir, também, com o serviço, recolhendo dados para integrar na monitorização.

A monitorização segundo esta forma apresenta, no entanto, uma camada de processamento adicional nos sistemas. O facto de se passar para uma plataforma *Java* acrescenta alguma entropia no próprio sistema monitorizado e no monitor (que tem de usar canais próprios – *Java Gateway* do *Zabbix* – para contactar remotamente a JVM). Deve-se, portanto, refletir nos casos em que se quer aplicar monitorização usando *MBeans* em detrimento de, por exemplo, SNMP ou agentes, pois o processo de obtenção desses indicadores via *MBeans* pode não apresentar ganhos a qualquer nível.

Definiram-se alguns critérios que devem ser seguidos aquando desta decisão:

- Já existe forma de obter a informação em questão, nativamente, no monitor?

- Já existe(m) caso(s) onde se obtém/obtêm dados segundo esse canal?

  - Se sim, é possível obter, de uma só vez, os dados para todos os itens?

  - Se não, é mais fácil adicionar novo canal? É mais fácil configurar um *MBean*?

- É mais exigente, em termos de sistema num todo (rede, processamento, armazenamento) usar o novo canal ou permanecer no antigo?

O facto de o *Java* conseguir, virtualmente, resolver todos os problemas de obtenção de informação, não deve ser pretexto para implementar *MBeans* para todos os casos, principalmente porque não só o *Java* adiciona mais uma camada de abstração que custa recursos, mas também é vulnerável a ataques.

4.4.3 Utilidade do *MBeans* para monitorização dos recursos da Associação DNS.PT

Após análise cuidadosa dos recursos que se queria monitorizar e do *trade-off* de desempenho da monitorização por *MBeans*, chegou-se à conclusão que seria proveitoso monitorizar alguns serviços desta forma. Assim, os serviços a monitorizar seriam:

- Avaliador Técnico: O avaliador técnico é um serviço *Web* que é disponibilizado no *Website* da Associação DNS.PT para confirmação das configurações técnicas de um domínio. É uma ferramenta crucial para o desempenho dos objetivos da Associação DNS.PT e como tal, é necessário saber se a performance está dentro dos valores aceitáveis ou se está com problemas.

- *Whois*: o *whois* é um serviço crucial em qualquer *registry*. É usando este serviço que os utilizadores da *Internet* (com especial incidência nos *registrars*) podem saber as informações técnicas de DNS para certo domínio. É usado intensivamente pois oferece as informações necessárias para descobrir a quem pertence um domínio, com quem contactar e como está configurado. Este serviço recebe um nome de domínio e, caso exista, devolve as suas informações.

Posto isto, procedeu-se à definição das formas de obtenção das métricas de performance para cada um destes serviços. Sendo serviços *Web*, uma das características mais importantes a ter em conta é o tempo que um utilizador normal demora a fazer uma utilização normal deste serviço. Entenda-se com esta descrição, a latência de resposta de um certo pedido. Assim, definiram-se alguns cenários para estes serviços:

Avaliador técnico – *Web Service*: Para este cenário, pretende-se obter as métricas de performance baseando-se num acesso *Web* via protocolo SOAP. Para medir a rapidez, dever-se-á seguir o URL do serviço, enviar o pacote SOAP para o servidor e esperar até que a resposta surja. Desta forma, será possível obter a latência de resposta que um utilizador normal tem.

Avaliador técnico – servidor: Neste cenário, pretende-se saber a latência que o próprio servidor tem a tratar o pedido. Este indicador permite discernir se eventuais atrasos nas respostas aos clientes advêm de problemas no servidor. Neste caso, a ligação é feita diretamente ao servidor interno que trata destes pedidos, e é enviado um pacote SOAP para este, medindo-se o tempo entre o envio do pedido e a receção da resposta. Desta forma é possível saber, efetivamente, o tempo que o servidor demora a tratar o pedido.

Whois – Web Service: Neste cenário pretende-se saber a latência no acesso ao serviço via *Web* no *browser*. Através do URL específico para um pedido simples (saber informações para o domínio *dns.pt*), recebe-se a resposta, analisa-se o *header* HTTP desta e verifica-se a existência de erros. Mede-se a latência entre o momento em que se abre a ligação com o URL e o momento em que se analisa o *header*. Posto isto, tem-se o tempo que um utilizador tem, aproximadamente, de esperar pela resposta via *Web*.

Whois – servidor: Este cenário serve para saber se uma eventual lentidão do serviço se deve ao servidor ou a fatores externos, como rede ou mesmo problema do cliente.

Faz-se a ligação ao servidor diretamente à porta para o serviço *WhoIs* – porta 43. A latência é medida desde que se escreve o pedido no canal (*stream)* estabelecido, até à receção da resposta. Desta forma, consegue-se saber se a latência de resolução dos pedidos no servidor está dentro dos valores normais ou não.

4.4.4 Implementação dos *MBeans*

Uma vez definidos os cenários, passou-se à implementação dos recursos *MBean* para monitorização da performance dos serviços já descritos. A primeira tarefa foi implementar a interface para servir os propósitos estabelecidos. Para satisfazer os requisitos da especificação JMX, o nome escolhido para a interface foi *TestConnectionMBean*, que declara os seguintes métodos:

- *sayTestConnection*() : descreve o funcionamento e propósito destes *MBeans*. Faz parte da especificação JMX [12].

- *getTRLatencyViaServer*() : função que retorna a latência de resposta do avaliador técnico via ligação ao servidor.

- *getTRLatencyViaWeb*() : função que retorna a latência de resposta do avaliador técnico via *Web.*

- *getWILatencyViaSocket*() : função que retorna a latência de resposta do serviço *Whois* via servidor.

- *getWILatencyViaWeb*(): função que retorna a latência de resposta do serviço *Whois* via *Web*.

Definiu-se ainda uma função que se prende com o servidor de *MBean* e não com a monitorização, que é a função *terminateServer(),* usada para terminar a execução do servidor remotamente.

Uma vez definida a interface, passou-se à implementação da classe *TestConnection*, que implementa a interface *TestConnectionMBean*, e vai ser responsável por concretizar a obtenção das latências de resposta dos serviços nos diferentes cenários.

Em seguida, descrevem-se as opções de implementação de cada um dos métodos anteriormente apresentados.

void sayTestConnection()

Este método mostra apenas uma janela com a mensagem "`This class declares a set of methods that test and measure the time it takes for the`

connection setup to the Technical Reviewer and WhoIs services. This MBean shows latency, in milliseconds, of the Associação DNS.PT's *Web Services*". Esta breve descrição aparece num *pop-up* recorrendo à classe *JOptionPane*, que usa elementos da classe *Javax.swing*, classe responsável pelo desenho de elementos gráficos no ecrã.

getTRLatencyViaServer()

Neste método, a comunicação é feita enviando um pacote SOAP para o servidor com o pedido, conforme os passos seguintes:

1) Criação de uma *factory*, classe que cria objetos *SOAPConnection*, cujo nome é *SOAPConnectionFactory*. Com esta *factory* obtém-se uma instância que servirá de base para as ligações, envios e receções de informação pelo protocolo SOAP.
2) Envio do pacote SOAP com o pedido. Recorre-se ao método que a classe *SOAPConnection* oferece, o *call*, cujos argumentos são uma função (de *callback)* e um URL. O URL é o caminho para o serviço *Web* do avaliador técnico no servidor e a função de *callback* é uma função privada que encapsula a definição dos parâmetros do pacote SOAP. O seu único propósito é criar uma instância da classe *SOAPMessage*, vinda da *MessageFactory*, e preenchendo-a adicionando a informação necessária ao *NameSpace* usado nesta comunicação SOAP. Retorna a *SOAPMessage* que representa o pedido que se pretende fazer ao servidor. A função *call* fica então responsável por enviar a mensagem até ao *endpoint* (o servidor) e bloqueia até receber a resposta. O URL usado é o localizador do servidor interno, para que se possa testar a latência de comunicação num ambiente interno.
3) Mede-se o tempo de latência de resposta (como a função bloqueia, no momento em que se faz a diferença de tempos, a resposta já é conhecida).
4) Termina-se a ligação, invocando a função *close()* da classe *SOAPConnection.*.

O valor que esta função retorna é o tempo em milissegundos que demorou a executar a tarefa de pedir ao avaliador técnico para avaliar o domínio *dns.pt*. Este valor vai ser devolvido ao servidor de *MBeans*, que o guardará no histórico de execução do método, sendo assim possível obtê-lo para efeitos de monitorização.

void getTRLatencyViaWeb()

Este método é igual ao anterior (*getTRLatencyViaServer*()) exceto em dois pontos: o URL, que desta vez será para aceder ao *frontend Web* que trata dos pedidos vindos do *Website* que disponibiliza a ferramenta, e o pacote SOAP, que tem algumas diferenças face ao enviado na comunicação direta com o servidor.

Este novo pacote é um pouco mais complexo do que o usado no método anterior. Neste pacote SOAP é necessário definir-se dois *NameSpaces*, um para o URL do serviço (*NameSpace* "ser") e outro com o *NameSpace* da definição do *schema* XML (*NameSpace* "xsd"). OS URLs a usar são, respectivamente, dos serviços *Web* (e das especificações do XML. Quanto ao corpo da mensagem, é necessário definir algumas informações que não foram descritas na anterior opção: o tipo de pedido (*domainEvaluationRequest*), o ID do domínio a avaliar (*domainID*) e a opção de guardar ou não o resultado (*saveResult*).

Depois do preenchimento do pacote SOAP, o procedimento é igual ao método anterior: guarda-se o tempo que passou desde o envio até à receção das informações e termina-se a ligação.

Este método é importante para se saber a latência que um utilizador comum pode estar a experienciar. Permite, em conjunto com o *getTRLatencyViaServer*(), saber se o sistema está lento, se existem dificuldades a nível da rede ou se o utilizador tem problemas no seu lado da comunicação.

getWILatencyViaSocket()

À semelhança do que acontece para o avaliador técnico, pretende-se saber o tempo médio que o serviço *WhoIs* demora a responder a pedidos. Com este método, a medição é feita com base na máquina que trata dos pedidos, podendo assim saber a latência na origem.

A forma como aqui se mede, neste caso, a latência do serviço *WhoIs*, é simples: estabelece-se uma ligação via *Socket* ao porto 43, envia-se o domínio pretendido via *Socket* e, aquando da receção da resposta, calcula-se a diferença de tempo entre receção e envio do pedido. No final de tudo isto, termina-se a ligação fechando o *Socket*.

getWILatencyViaWeb()

Este método para obtenção da latência do serviço *WhoIs* é um pouco diferente do usado para o serviço avaliador técnico. O procedimento é de acordo com a seguinte descrição:

1) Começa-se por se criar um objeto "URL", que identifica um recurso na *Web*. Este URL vai permitir estabelecer uma ligação HTTP ao servidor do recurso pretendido.
2) Abre-se a ligação usando o método *openConnection(),* que executará o pedido ao serviço *WhoIs* de acordo com o que está especificado no URL para obtenção das informações do domínio "dns.pt".
3) Obtém-se o *statusCode* da resposta via *header* HTTP da resposta, usando o método *getHeaderField(0).* Caso a resposta seja *OK* (*statusCode* > 200 & <

300), devolve-se o tempo que demorou a ação a decorrer. Caso contrário, um erro é lançado a indicar que não houve resposta válida.

Este método vai permitir saber se existem tempos de espera fora do normal para o serviço de *WhoIs*, aquando da utilização do serviço *Web* para obter informações acerca de um domínio. Em conjunto com o método *getWILatencyViaSocket()*, é possível ter uma noção do estado do serviço *WhoIs*.

terminateServer()

O último método desta classe é o que permite, remotamente, terminar o servidor de *MBeans*, caso seja necessário. A sua implementação é a seguinte:

1) Aquando da invocação, é apresentada um *pop-up* com a informação que o servidor irá ser encerrado.
2) A execução do programa servidor *MBeans* é terminada, enviando um pedido de *System.exit(0)*, sendo que o 0 indica que terminou pacificamente.

4.4.5 *Mbeans* no plano de monitorização

Os *MBeans* foram pensados como uma parte da solução de monitorização e, como tal, seriam incluídos na plataforma *Zabbix* para que se pudesse ter não só monitorização das latências de comunicação, mas também poder gerar alarmística com base nos dados obtidos. No entanto, antes de se integrar com a plataforma de monitorização, os seguintes pontos tiveram de ser ponderados:

1) Qual o melhor ponto na rede para realizar os testes?
2) Quer-se monitorizar as conexões internas e externas aos serviços a partir do mesmo ponto de origem?

Pode considerar-se que o ponto 2) é consequência do ponto 1), pois para escolher o melhor ponto para realizar os testes às ligações é necessário saber se se pretende ter uma separação ou se se quer testar em bloco a partir de um ponto específico.

Testar na rede interna traz alguma veracidade e coerência aos resultados das ligações feitas diretamente aos servidores que hospedam os serviços, mas descura-se a componente principal das ligações aos serviços via *Web*, que é aceder aos serviços pelos meios que o público em geral acede. Por outro lado, monitorizar a partir de um ponto exterior apresenta o oposto: o objetivo de testar as ligações diretas aos servidores é deturpado pois não se consegue discernir problemas de serviços ou problemas terceiros (sendo a ligação de rede a mais importante).

Chega-se, portanto, à conclusão que o mais coerente será ter dois perfis para a monitorização: um público, onde, de um ponto externo à rede dos servidores e da infra-estrutura técnica (*firewalls*, principalmente), se testam as ligações via *Web*; e um interno, de onde se monitoriza o funcionamento do serviço a partir da mesma rede e infra-estrutura técnica onde se situam os servidores visados, permitindo assim perceber, aquando de valores anómalos vindos de qualquer das partes, se o problema vem de causas exteriores (onde a latência interna se mantém normal e a latência externa é elevada) ou de causas interiores à infra-estrutura (latência interna e externa elevadas).

4.4.6 Instalação e Configuração

No lado do servidor que irá hospedar o servidor de *MBeans*, apenas é necessário executar o programa *Java* e abrir a porta para as ligações JMX para que o servidor de monitorização possa aceder aos dados. Essa porta é definida *a priori* e, para este cenário, escolheu-se a porta 9999, muitas vezes usada como *default* para o efeito.

Numa primeira fase, escolheu-se não ativar os mecanismos de segurança para proteger a ligação JMX. Isto deve-se ao facto de, aquando da implementação, tudo foi feito sem acrescentar a camada de segurança à ligação para facilitar os testes e porque, a nível interno, as ligações estão protegidas com uma *firewall* institucional. Quanto à ligação com o servidor *MBeans* de perfil público, será acrescentada, então, a camada de segurança, adicionando uma palavra-passe e encriptando a ligação, recorrendo aos mecanismos nativos ao JMX para o efeito.

Quanto ao servidor de monitorização, apenas se terá de configurar um novo conjunto de *Items*, que serão obtidos via *Java Gateway*. Os itens recebidos irão conter a latência em milissegundos (conforme a especificação) e serão definidos *Triggers* de acordo com os valores que se acharem anómalos para as latências de resposta dos serviços em questão.

As Figuras 25 e 26 ilustram a forma de utilizar os dados de monitorização dos *Web Services* para obter uma representação gráfica. Embora não se preveja usar o *Java Mission Control* para as monitorizar, a existência de *dials* (ponteiros que vão movendo lateralmente para indicar aumento e diminuição das latências dos serviços) fornece uma forma de visualização muito útil para este tipo de dados.

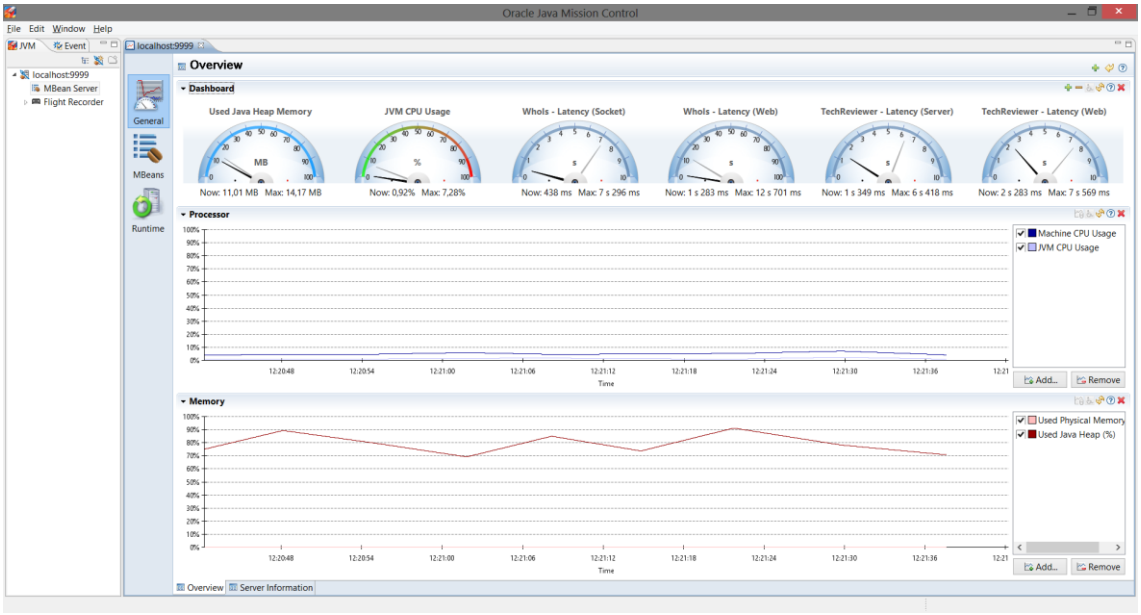


Figura 25 - Ambiente de testes: Interface do Java *Mission Control*, plataforma que permite consultar os *MBeans*. Na figura, o *Dashboard* com os *dials* a indicarem, em tempo-real, a latência dos serviços.

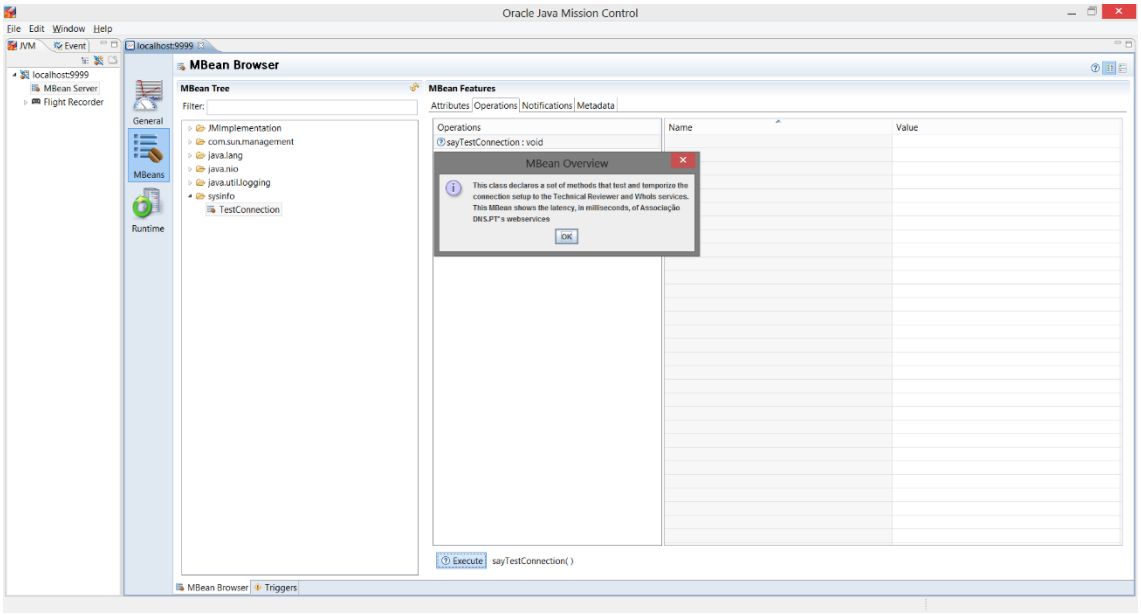
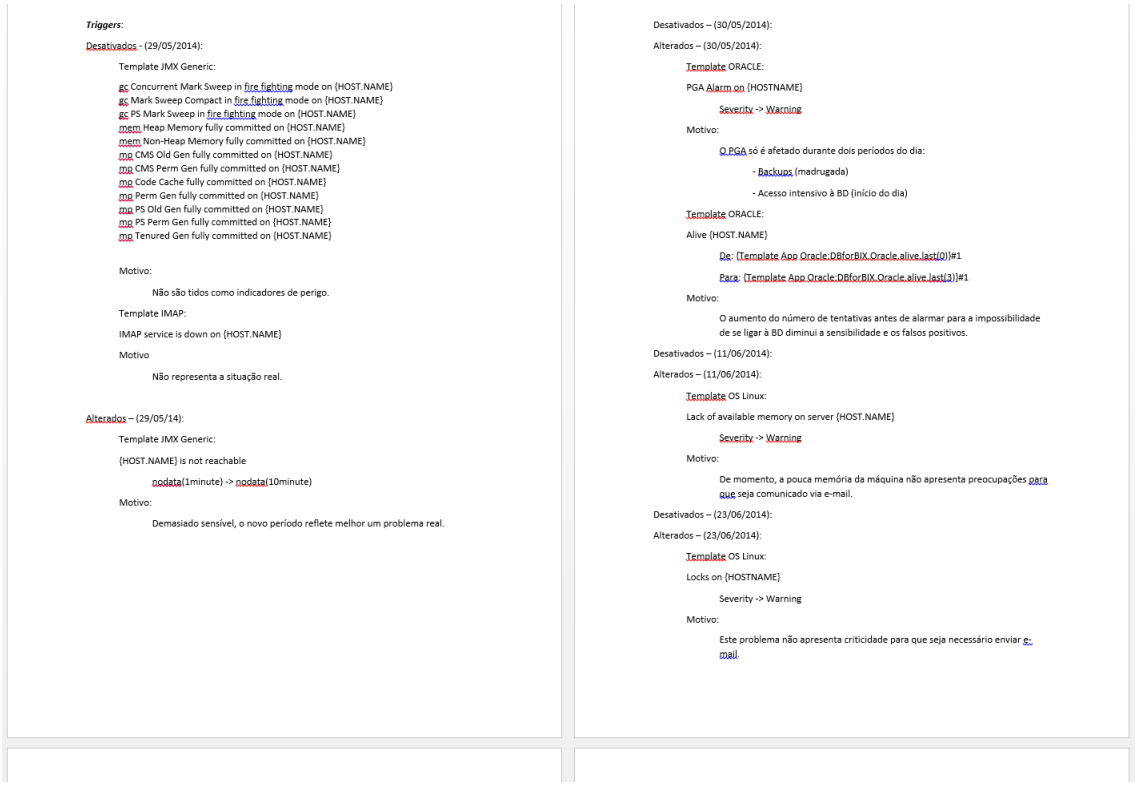


Figura 26 - Ambiente de testes: A mensagem que aparece quando se invoca o método *sayConnectioTest()*.

4.5 Estado Atual do Sistema de Monitorização

Após todas as ações efetuadas de criação e configuração das componentes do novo sistema de monitorização, foi tempo de avaliar o trabalho feito. Esta avaliação começou por olhar para o sistema agora funcional e fazer dois julgamentos: o que há e o que falta.

Perante o acompanhamento à evolução do sistema feito diariamente, iam-se efetuando alguns ajustes, como alterações nas sensibilidades dos *Triggers* definidos ou alterações nos tipos de indicadores recebidos, nomeadamente a nível das periocidades. Durante algum tempo, manteve-se um diário de bordo para as alterações que foram sendo feitas, de acordo com o que a Figura 27 ilustra.



Triggers:

Desativados - (29/05/2014):

Template JMX Generic:

gc Concurrent Mark Sweep in fire fighting mode on {HOST.NAME}
gc Mark Sweep Compact in fire fighting mode on {HOST.NAME}
gc PS Mark Sweep in fire fighting mode on {HOST.NAME}
mem Heap Memory fully committed on {HOST.NAME}
mem Non-Heap Memory fully committed on {HOST.NAME}
mp CMS Old Gen fully committed on {HOST.NAME}
mp CMS Perm Gen fully committed on {HOST.NAME}
mp Code Cache fully committed on {HOST.NAME}
mp Perm Gen fully committed on {HOST.NAME}
mp PS Old Gen fully committed on {HOST.NAME}
mp PS Perm Gen fully committed on {HOST.NAME}
mp Tenured Gen fully committed on {HOST.NAME}

Motivo:

Não são tidos como indicadores de perigo.

Template IMAP:

IMAP service is down on {HOST.NAME}

Motivo

Não representa a situação real.

Alterados – (29/05/14):

Template JMX Generic:

{HOST.NAME} is not reachable

nodata(1minute) -> nodata(10minute)

Motivo:

Demasiado sensível, o novo período reflete melhor um problema real.

Desativados – (30/05/2014):

Alterados – (30/05/2014):

Template ORACLE:

PGA Alarm on {HOSTNAME}

Severity -> Warning

Motivo:

O PGA só é afetado durante dois períodos do dia:

- Backups (madrugada)
- Acesso intensivo à BD (início do dia)

Template ORACLE:

Alive {HOST.NAME}

De: {Template App Oracle:DBforBIX.Oracle.alive.last(0)}#1

Para: {Template App Oracle:DBforBIX.Oracle.alive.last(3)}#1

Motivo:

O aumento do número de tentativas antes de alarmar para a impossibilidade de se ligar à BD diminui a sensibilidade e os falsos positivos.

Desativados – (11/06/2014):

Alterados – (11/06/2014):

Template OS Linux:

Lack of available memory on server {HOST.NAME}

Severity -> Warning

Motivo:

De momento, a pouca memória da máquina não apresenta preocupações para que seja comunicado via e-mail.

Desativados – (23/06/2014):

Alterados – (23/06/2014):

Template OS Linux:

Locks on {HOSTNAME}

Severity -> Warning

Motivo:

Este problema não apresenta criticidade para que seja necessário enviar e-mail.

Figura 27 – Excerto do documento com as afinações do sistema de monitorização.

Esta avaliação que tem sido levada a cabo desde o dia em que o sistema ficou operacional, proporcionou a adaptação do sistema à realidade da Associação e às suas necessidades. Este acompanhamento constante permitiu identificar lacunas na monitorização, o que levou ao planeamento de nova monitorização e alarmística, fora do pensado inicialmente.

Identificar o que faltava fazer veio naturalmente com o tempo. De facto, dizer que um sistema deste cariz está "fechado" e que a implementação foi terminada não é um cenário realista. Da utilização diária apercebe-se que há novos indicadores que são úteis

recolher, ou que certos indicadores ficaram obsoletos com a introdução de outros mais adequados. Ao longo do tempo percebe-se que a alarmística pensada numa fase inicial não está perfeitamente adequada ao estado do sistema atual e é necessário abordar o caso de outra forma. É, portanto, natural que tenham sido identificados novos objectivos mal o sistema de monitorização terminou a fase de configuração dos aspetos críticos.

Nesta segunda iteração de configurações abordou-se a monitorização e alarmística para sistemas que não eram críticos na fase inicial do sistema, mas cuja importância ditou que fossem integrados no sistema de monitorização o quanto antes. De seguida, listam-se os sistemas cuja monitorização foi integrada nesta fase:

- Monitorização de mais aspetos do *BIND* (*SerialCheck* e *QueryTime*)
- Monitorização DNSSEC
- Monitorização do disco das máquinas
- Monitorização do sistema DSC

4.5.1 - Monitorização *BIND*

A monitorização do *core* do serviço *BIND* foi alcançada durante a primeira fase de implementação. Depois disso, alguns indicadores úteis que foram adiados na altura da primeira iteração foram revisitados.

A monitorização em falta prendia-se com a verificação do *Serial Number* da instalação da zona (cada instalação de uma nova zona tem o *Serial Number* diferente, para distinguir zonas ao longo do tempo) e a obtenção da latência de comunicação que se tem para com os servidores de nomes primário e secundários de certa zona. Estes indicadores permitem, de uma forma ainda mais alargada, perceber se o sistema *BIND* está a funcionar corretamente.

Para a monitorização das latências de comunicação, criou-se um *script* simples que verifica quanto tempo demorou a obter um *resource record* de um certo servidor de nomes. Esse tempo é posteriormente enviado para o sistema de monitorização, permitindo verificar se um certo servidor de nomes está a demorar mais tempo que o habitual a responder.

A monitorização dos *Serial Numbers* teve uma grande importância, quer para o conhecimento mais aprofundado do estado do *BIND*, quer para o enriquecimento do conhecimento sobre as valências do sistema de monitorização, devido à forma que se encontrou para implementar a obtenção destes indicadores.

O grande problema que levou a que se procurasse uma nova forma de integrar esta informação no sistema de monitorização foi a impossibilidade de configurar um *Item* para ser recolhido a certas alturas do dia, por exemplo, de duas em duas horas, mas sempre na hora certa (às 12h00, 14h00, 16h00, etc.). Como os *Items* são periodicamente obtidos com base na data em que se definiu essa periocidade (ou seja, se se definiu uma periocidade de duas horas às 12h45, o indicador será recolhido às 14:45, 16:45, 18h45, etc.), não seria fácil fazer com que um *Item* fosse obtido nas horas certas.

Para contornar este problema, descobriu-se que o *Zabbix* permite subverter a lógica de integração de dados no sistema de monitorização, permitindo que seja o agente a enviar os dados para o servidor, periodicamente. Esta forma utiliza o *zabbix_trapper* (porto 10051) na comunicação com o servidor. Este método foi conciliado com um *script* e com o conceito de calendarização de execuções de *scripts*, de forma a conseguir o envio automático e periódico da informação para o servidor.

O objetivo desta monitorização é saber o atraso que um certo servidor secundário tem em relação ao servidor primário, para a instalação de uma zona. Isto é importante porque, para a sanidade do sistema de DNS, é importante que não haja servidores de nomes a difundir informações díspares para a mesma zona.

Quanto ao *script* em si, o seu funcionamento é o seguinte:

- Obtém-se a informação do *Serial Number* para o servidor primário da zona em questão.

- Obtém-se a listagem de servidores de nomes secundários para a referida zona.

- Para cada servidor de nomes secundário faz-se a comparação com o *Serial Number* do servidor de nomes primário. Se a verificação falhar, envia-se para o servidor de moniotirzação um valor a indicar que os *Serial Numbers* não são iguais; caso contrário, envia-se a confirmação de que está tudo bem.

Este *script* é configurado no *crontab* da máquina (aplicação que gere a execução automatizada de comandos, baseando-se em períodos temporais pré-configurados) para ser executada todas as horas pares, uma vez que a instalação da zona .pt ocorre todas as horas ímpares do dia, contabilizando uma hora para as alterações se propagarem pelos servidores de nomes secundários, que recebem a notificação para atualizarem a sua zona a diferentes momentos.

Em termos de configuração no sistema *Zabbix*, as diferenças entre um *Item* usando a comunicação normal com o agente *Zabbix* e na forma *zabbix_trapper* consiste, principalmente, na não especificação da periocidade de atualização de dados. Em vez disso, tem-se de inserir a chave do *Item*, para que o servidor *Zabbix*, quando receber a

actualização, associe o tempo e o *Host* aos dados recebidos. A Figura 28 mostra como configurar um *Item* com *zabbix_trapper* na interface *Web* do *Zabbix*.

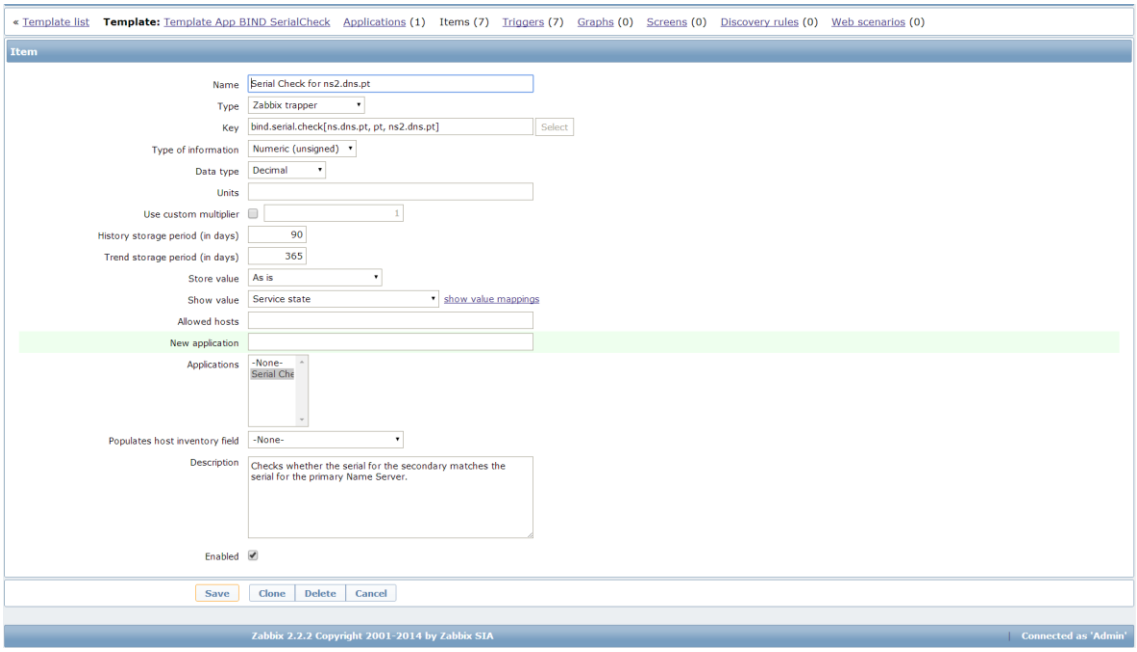


Figura 28 – Ecrã de configuração de um *Item* de *Serial Check*, usando o *zabbix_trapper*

4.5.2 - Monitorização DNSSEC

DNSSEC é um conjunto de extensões que foram introduzidas via *resource records*, assinaturas e chaves digitais para aumentar a segurança do protocolo DNS, e assim ajudar na crescente necessidade dos sistemas se protegerem perante atos maliciosos ou suspeitos que ocorrem ao nível deste protocolo, permitindo adicionar uma camada de autenticidade ao tráfego DNS [17]. O DNSSEC contribui para segurança informática no atual paradigma de utilização segura da Internet, pelo que monitorizar o seu funcionamento torna-se imprescindível.

A monitorização é feita através da verificação da validade das zonas que são assinadas com DNSSEC (verifica-se se a data de expiração da assinatura já passou). Para tal concebeu-se um *script* que faz a seguinte verificação:

- Obtém-se a data de assinatura da zona pretendida (por exemplo, zona .pt) pedindo ao servidor de nomes pelo *resource record* DNSKEY, que indica a chave pública usada no processo criptográfico da assinatura. Este *resource record* contém um campo para a data de assinatura, em formato "*anomêsdiahoraminutosegundo*".

- Converte-se essa data para formato *UNIX Epoch*, que traduz uma data em segundos passados desde 1 de Janeiro de 1970 até à data atual.

- Faz-se a diferença entre a data atual (em formato *UNIX Epoch*) e a data de assinatura, obtendo o número de segundos passados desde a assinatura.

- Converte-se esse resultado em dias

(- Envia-se o valor para o servidor de monitorização.)

Com esta informação, faz-se a monitorização da validade de uma zona, sendo que, para diferentes zonas, existem diferentes periocidades para a assinatura (a zona .pt é assinada de duas em duas horas, enquanto as restantes zonas relevantes para monitorização são assinadas duas vezes por mês). No último passo, usado apenas para a zona .pt, usa-se a comunicação *zabbix_trapper* para enviar os dados para o servidor cinco minutos após o início das horas ímpares. Para as restantes zonas, manteve-se a forma normal, deixando o servidor recolher os dados junto do agente.

A Figura 29 representa a configuração, na interface *Zabbix*, da validade da assinatura da zona .pt, que usa o *zabbix_trapper*.

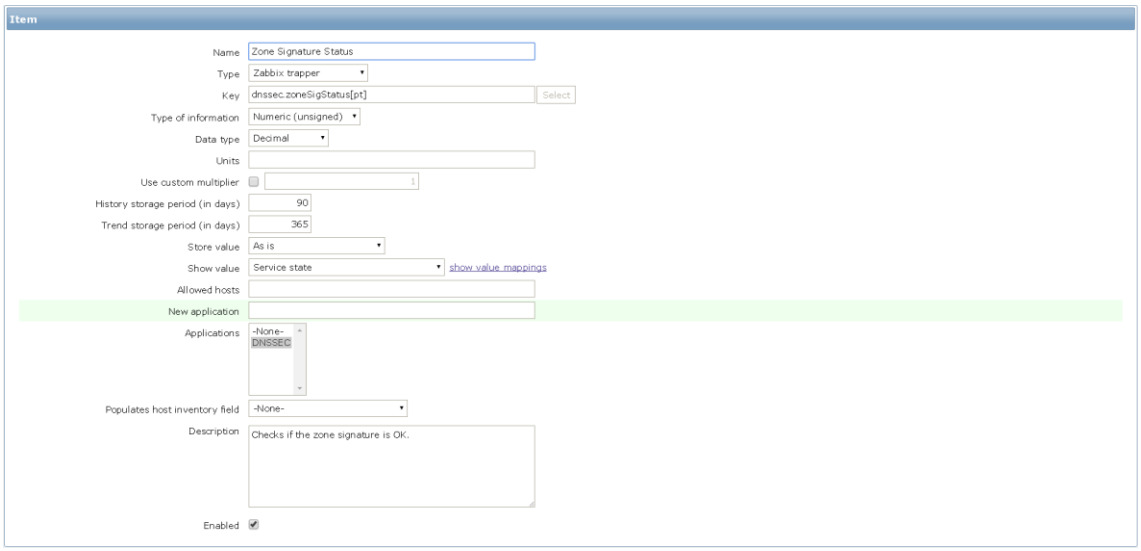


Figura 29 – Ecrã de configuração do *Item* para monitorização da assinatura da zona PT

4.5.3 - Monitorização do Disco das Máquinas

O disco rígido é um recurso precioso para o bom funcionamento de qualquer servidor. Para saber o seu estado, descobriu-se que o sistema operativo *Linux* disponibiliza um ficheiro – *diskstats* – que reúne os tempos que os vários tipos de operações de disco estão a demorar. Esta informação permite saber, por exemplo, o

tempo que o disco está a executar leituras, escritas ou em *I/O* (tempo que se passou à espera que se acedesse ao disco para executar uma operação).

Para implementar esta monitorização utilizou-se uma forma proposta por *Dennis Kanbier* no seu *blog* [18], que utiliza o *diskstats*. O conteúdo do ficheiro é filtrado e é obtenível através de um conjunto de *UserParameter* definidos no *Template* que *Dennis* criou.

Na integração com o *Zabbix* usa-se o *Template* facultado para definir uma forma diferente de configurar os *Items* – os *Discovery Items*. O *Discovery* é uma fase na comunicação com os *Hosts* em que são estes que indicam o que se pode obter do sistema monitorizado.

Para usar um *Discovery Item*, configuram-se protótipos (*Prototypes* na aplicação *Zabbix*, ver Figuras 30 e 31) que têm um esqueleto, uma abstração daquilo que vai ser feito. Depois, é necessário que, no *Host*, haja algo que forneça a informação que irá despoletar a criação dos *Items*, *Triggers* e outros *Prototypes* definidos.

Para concretizar esta monitorização, cria-se um *script* que forneça a informação dos discos (nomes, principalmente) presentes na máquina, num formato que o *Zabbix* entenda (neste caso, é usada a notação JSON) para, na fase de *Discovery*, executar e obter o resultado. Com essa informação recolhida, as variáveis nos protótipos são substituídas por informação concreta, permitindo assim que se criem *Items*, *Triggers* e *Graphs* com correspondência a informação real.

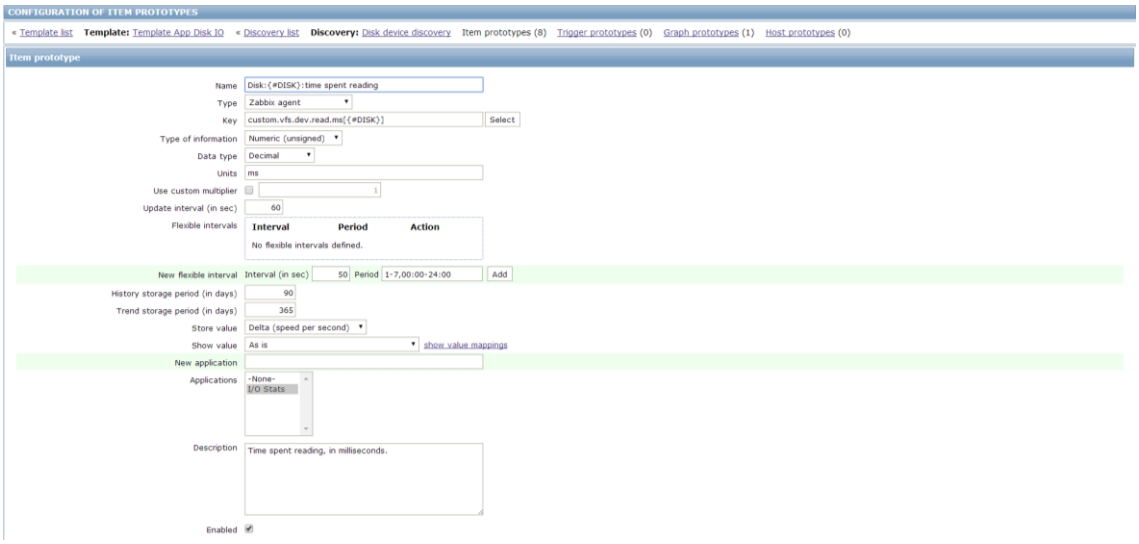


Figura 30 – Ecrã de configuração de um *Item Prototype*, do *Discovery Item*

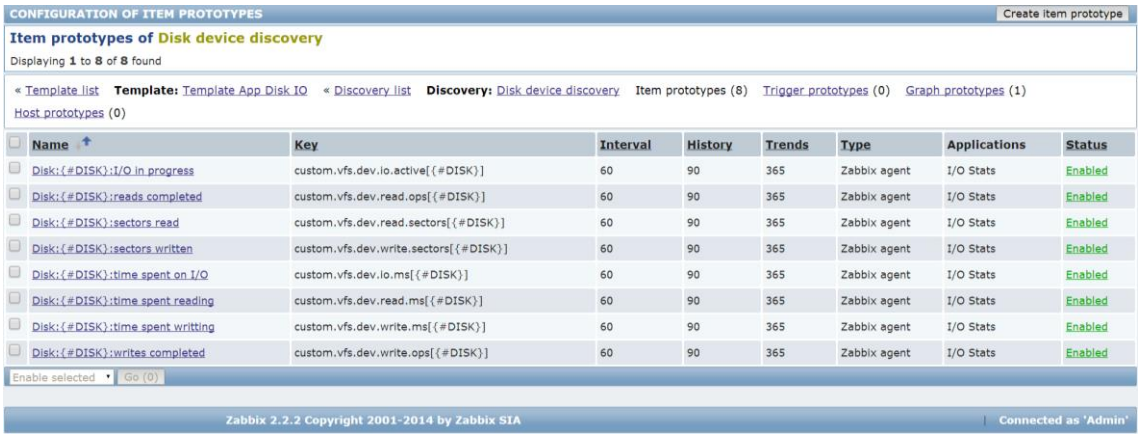


Figura 31 – Ecrã com a lista de *Item Prototypes* que darão lugar a *Items* nos *Hosts*

Depois de criados os *Items* com a informação toda, são usados os *UserParameter* que irão usar o ficheiro *diskstats* para recolher a informação que será associada, em cada *Host*, para as várias partições encontradas. Esta capacidade que os *Discovery Items* oferecem de não se prender a configurações estáticas oferece ao sistema de monitorização uma dinâmica muito interessante, permitindo que seja o próprio sistema a descobrir que *Items* irá criar a partir de protótipos e de listagens que o *Host* disponibiliza.

4.5.4 - Monitorização do sistema DSC

O sistema DSC tem capacidade de captar informação que o sistema de monitorização *Zabbix* e os *scripts* de recolha de indicadores do *BIND* não têm. No entanto, como não é possível integrar diretamente os dados do DSC no *Zabbix*, o que se consegue monitorizar para este sistema é relativamente limitado.

A monitorização do DSC que é feita no sistema *Zabbix* é quanto ao estado de execução da aplicação DSC nas máquinas *collector* e quanto ao espaço em disco usado pelo DSC na máquina *presenter*.

No que toca a saber se o DSC está a correr ou não, usa-se uma forma análoga à que se usou na monitorização do *Apache* (método "a)" no Capítulo 4, secção 4.3.3), executando-se o comando remoto na máquina *collector* e vendo se existe alguma tarefa com o nome DSC. A alarmística neste caso é o despoletar de um aviso caso não exista nenhuma tarefa a executar com o nome DSC.

Quanto à ocupação do disco do *presenter*, a monitorização necessitou de recorrer a *scripts*, pois não é possível obter o espaço que dada diretoria ocupa no disco, recorrendo

apenas às funções nativas do *Zabbix*. Como o espaço em disco é um recurso finito e que, chegando a níveis muito baixos, faz com que a aplicação DSC (e outras aplicações) deixe de funcionar corretamente, a criação de uma forma avançada de monitorização foi uma necessidade.

Para concretizar esta monitorização configuraram-se *UserParameters* no *presenter* do DSC que fazem a leitura de um ficheiro que é periodicamente atualizado com o espaço que cada diretoria de *collector* está a usar. No lado do servidor, existe um *Graph* com a representação do espaço que cada *collector* está a usar no *presenter*, como se pode ver na Figura 32.

Desta forma, sabe-se a qualquer momento quanto é que o volume de dados do DSC está a ocupar, permitido tomar medidas caso tenha um crescimento fora do normal (que acima, quer abaixo das expetativas).

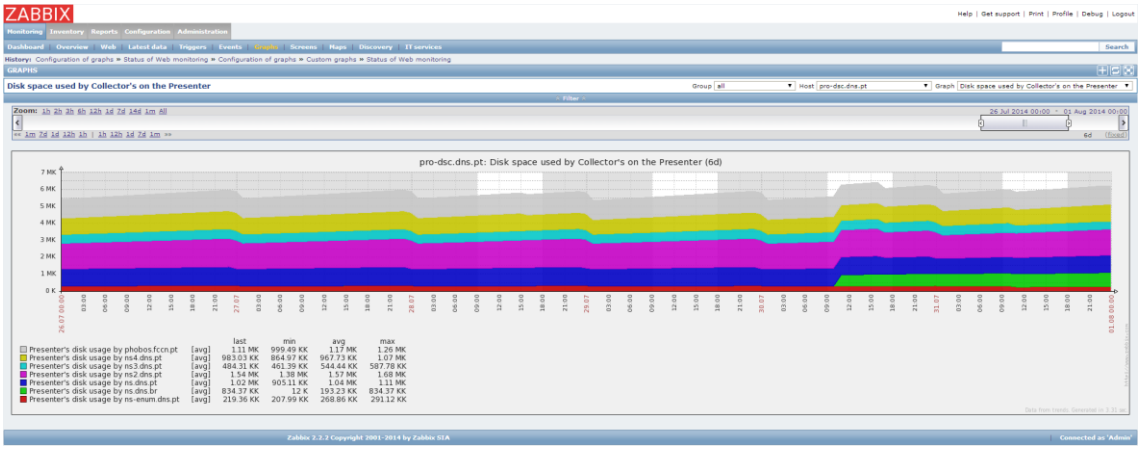


Figura 32 – Ecrã de *Graph* que representa o espaço usado pelos *Collectors* no *Presenter* DSC

Capítulo 5 *Dashboard DNS.PT*

5.1 Contextualização

A necessidade de monitorizar os recursos da Associação foi satisfeita com a implementação do sistema de monitorização *Zabbix*. No entanto, um dos aspetos fulcrais de um sistema de monitorização é a forma como a informação é apresentada, admitindo uma importância acrescida se se tiver em conta que se tem de conhecer os vários estados dos sistemas de uma forma prática e que imediata. Desta forma, um bom sistema de monitorização e alarmística deve, não só, ter um bom sistema de obtenção de indicadores, mas também ter uma boa interface gráfica que permita uma representação dos indicadores e da alarmística associada a estes, intuitiva na utilização e versátil naquilo que consegue disponibilizar.

5.2 Levantamento de Requisitos

O *dashboard* por defeito da plataforma *Zabbix* fica aquém das necessidades da equipa técnica, que lida diariamente com a leitura e interpretação dos vários dados que são disponibilizados pelo sistema de monitorização. Esta interface foi usada intensivamente (não só o *dashboard* mas também outras páginas de gráficos e tabelas dos indicadores recebidos) aquando do sistema antigo de monitorização.

Com o desenho e implementação de um novo sistema de monitorização, achou-se oportuno, para satisfazer as necessidades da equipa técnica, levantarem-se alguns requisitos mínimos que a interface deveria ter:

- Deve ser intuitiva: Manusear a interface deve ser simples e percetível. Não deve haver ambiguidade de interpretação nas operações que a interface apresenta, e as operações que a interface consegue desempenhar devem ser facilmente reconhecíveis no mapeamento das mesmas (deve ter mnemónicas que clarifiquem o que cada acção fará).

- Deve ser completa: O *dashboard* deve apresentar a informação mais relevante seguindo o *princípio do Pareto* [19]: 80% dos dados que se visualizam devem corresponder a 20% de toda a infra-estrutura. Nesses 20% devem estar os aspetos mais

críticos e recorrentes da infra-estrutura técnica, aqueles que, habitualmente, necessitam de maior atenção. Ao aplicar este princípio, deve-se ter uma interface que seja capaz de, habilmente, representar informações suficientes para detetar os casos mais reincidentes. No entanto, o *dashboard* deve ser capaz de apresentar informação que alerte para problemas com menos incidência mas que nem por isso têm menos criticidade.

- Deve ser dinâmica: a informação disponibilizada no *dashboard* deve ser atualizada periodicamente e sem a intervenção do utilizador. Desta forma, a informação deve ser apresentada na interface e atualizações que ocorram no sistema de monitorização, devem fazer-se refletir.

Existe ainda um outro requisito, relacionado com a forma de apresentação, que se prende com a existência de um ecrã gigante na área técnica: a área de visualização do ecrã principal do *dashboard* deve ocupar um ecrã, sem ter conteúdo apenas visualizável fora do *viewport* (área de desenho da página), excluindo-se os casos de *dropdowns* e outros estilos de apresentação de conteúdos propositadamente escondidos. Desta forma, a informação está sempre totalmente visível no ecrã gigante.

- Deve ser apelativa: os problemas devem ser facilmente detetados e entre a informação visual apresentada e a existência de técnicas de representação de dados – como gráficos, tabelas e outros estilos de representação de dados – é altamente encorajada para mais facilmente passar informação e nela discernir eventuais problemas.

Com este levantamento de requisitos passou, então, a existir um conjunto de critérios que permitiriam fazer um julgamento ponderado de que interface seria utilizada para o novo sistema de monitorização.

5.3 *Dashboard Zabbix* – Avaliação

A interface gráfica do *Zabbix* (Figura 33) foi posta em causa depois do levantamento de requisitos. Perante todos os critérios levantados, fez-se uma apreciação daquilo a que a interface correspondia e não correspondia. Apresenta-se, de seguida, a avaliação feita ao *dashboard* da plataforma *Zabbix*:

- Interface intuitiva: A interface do *dashboard Zabbix* é uma interface simples e algo minimalista. Composta por apenas alguns painéis informativos, não tem problemas de interpretação daquilo que oferece a nível de interoperabilidade. No entanto, e ainda que com um conjunto algo minimalista das informações que apresenta, registam-se alguns problemas:

1. Existem diversos painéis com a indicação dos problemas que cada *Hostgroup* tem, a dado momento. Quando se desloca o cursor sobre cada um destes, aparece uma janela que identifica, para cada um, que *Host* tem problemas e a descrição do problema, bem como um *timestamp*. A janela em questão desaparece assim que se desloca o cursor, impedindo de se navegar para a página com mais informações acerca do problema em questão. Como o cursor não muda para o cursor de *hyperlink* (usando a mnemónica do punho com o indicador esticado) não se percebe que, clicando, a janela fica fixa, permitindo, assim, a navegação pelos *hyperlinks*.

2. A informação está dispersa por vários painéis. O painel de *System Status* contém colunas para cada severidade de problema (em ordem crescente de severidade, *Not Classified, Informaion, Warning, Average, High e Disaster*), e linhas para cada *Hostgroup* definido. Para cada linha, dada coluna tem o número de ocorrências para os *Hosts* desse grupo. Apenas deslocando o cursor se consegue saber quais os problemas que cada *host* tem. Já no painel de *Host Status*, repetem-se as linhas com *Hostgroups*, tendo-se agora três colunas, uma com o número de *Hosts* com problemas, outra com o número de *hosts* sem problemas e uma coluna com o total de *Hosts* para cada *Hostgroup*. Para além da duplicação de informação (o painel acima descrito oferece toda esta informação e mais alguns detalhes), ao se deslocar o cursor para a célula com indicação de problemas, apenas é apresentado uma tabela de *Hosts* por severidade do problema, tendo apenas os números de problemas para cada severidade, sem ser navegável.

- Interface completa: Quanto à completude da informação que é apresentada, não existem problemas a identificar: toda a informação relevante quanto aos problemas no sistema é visualizável a parir do *dashboard*. É possível saber que *Hosts* têm problemas, que problemas são e quando foram identificados. Para visualizar mais informações é possível navegar pela interface e aceder a outras páginas com a informação mais detalhada para os *Hosts*, problemas totais ou individuais para cada componente monitorizada, e visualizar gráficos e tabelas de indicadores recolhidos.

- Interface dinâmica: As informações presentes no *dashboard* vão sendo atualizadas, por períodos configuráveis (existe, em cada painel, um botão de menu que permite definir o intervalo temporal de atualização) e cada painel tem, no rodapé, o *timstamp* da última atualização.

- Interface apelativa: Neste aspeto, a interface *dashboad Zabbix* falha. Embora apresente cores diferentes consoante os diferentes problemas e severidades que são encontrados, o grafismo é, em termos gerais, pobre: o tamanho de letra demasiado pequeno para ser facilmente percetível à distância, não tem gráficos integrados nesse

mesmo *dashboard*, e a disposição dos painéis, que embora amovíveis, gera sempre organizações que não tiram o máximo proveito do espaço de visualização da página.

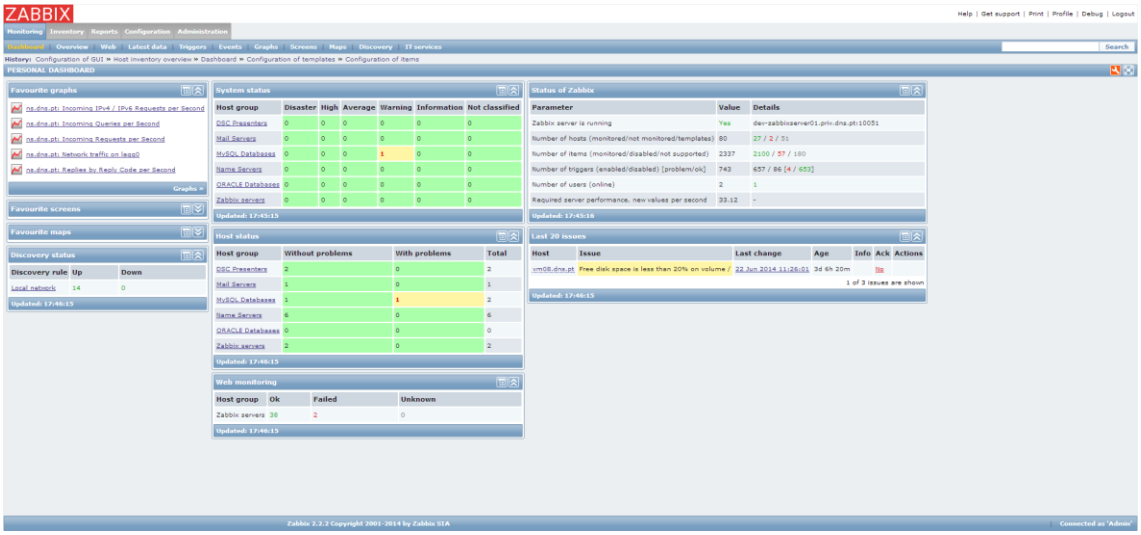


Figura 33 – Exemplo de uma configuração do *dashboard Zabbix*.

5.4 *Dashboard Zabbix* – Veredito

Após a análise crítica feita à interface do *dashboard* da plataforma *Zabbix*, percebeu-se que existiam alguns pontos que se pretendiam ter numa interface *dashboard* para o novo sistema de monitorização e que o *dashboard Zabbix* não contemplava. Embora cumpra os requisitos da completude e dinamismo da informação, o facto de ter alguma repetição da informação que apresenta e de ser muito orientada à interação com o utilizador, achou-se que o melhor seria optar por uma interface que apresentasse tudo o que se queria ver (mantendo o *princípio do Pareto*), com um fluxo de ações simples para chegar da visão macroscópica do sistema até a um detalhe de um *Host*. Adicionalmente, seria imprescindível que a interface apresentasse sempre a informação o mais atualizada possível, sem que para isso fosse necessária intervenção do utilizador, utilizando, de preferência, toda a área de visualização para apresentar a informação.

Concluiu-se que a interface para o *dashboard* do novo sistema de monitorização seria algo mais adaptado às necessidades específicas da equipa técnica, pelo que se teria de optar entre procurar, na comunidade de utilizadores da plataforma *Zabbix*, uma interface que assegurasse os requisitos levantados e que fosse compatível com a versão da plataforma *Zabbix* utilizada; ou desenvolver uma nova interface à medida das necessidades e requisitos levantados.

5.5 Levantamento de *Dashboards* da Comunidade

Com base no que se procurava num *dashboard*, procedeu-se à procura, entre a comunidade de utilizadores da plataforma *Zabbix*, de interfaces feitas por utilizadores, *open source* (livres para utilizar segundo a permissão do autor) e que integrassem bem com o sistema *Zabbix* na versão utilizada.

A procura não foi frutífera: a grande parte das aplicações encontradas limitavam-se a mudar os tipos de painéis que são apresentados no *dashboard*, como a interface da Figura 34, que apenas apresentava mais informação nos painéis do *dashboard*. Embora mais úteis que os que são disponibilizados pela própria plataforma *Zabbix*, apenas apresentavam uma melhor organização e possibilidade de configurar que indicadores apareceriam em cada painel ou simplesmente uma expansão da informação que, na versão original, estava acessível apenas depois de passar com o cursor sobre painéis. Se bem que esta versão seria mais útil à equipa técnica do que a versão usada se apresentava, faltava uma componente importante: poder integrar os gráficos com os alarmes que vão sendo gerados à medida que os dados de monitorização vão sendo analisados.

Esta componente gráfica assume especial importância pois é mais fácil analisar um gráfico no ecrã do que conjuntos de dados em tabelas. Além dos alertas sobre aquilo que está a acontecer, é também útil que haja apresentação da evolução do estado de certos sistemas para poder haver proatividade na ação preventiva de incidentes.

Figura 34 – Exemplo de um *Dashboard* customizado, apresentado pela comunidade. [20]

Após concluir que a comunidade não apresentava opções viáveis para implementar uma nova interface para o *dashboard*, decidiu-se avançar com a implementação de uma nova interface, construída de raiz, que satisfizesse todas as suas necessidades, de acordo com os requisitos levantados e apoiado pelas solicitações e sugestões de toda a equipa técnica.

5.6 Novo *Dashboard* DNS.PT

Depois da pesquisa sem resultados por um *dashboard*, foi necessário definir um plano para delinear o processo de desenvolvimento da interface pretendida. Este plano começou com uma fase de descoberta de requisitos, funcionais e não-funcionais, para a nova interface, bem como que tecnologias a usar, de acordo com a seguinte lista:

Aspetos Tecnológicos:

- Linguagem a usar: *Java* / PHP

- Obtenção da informação: acesso à base de dados *Zabbix* / API *Zabbix* / Outro

- Tipo de interface: HTML + *JavaScript* ou HTML + *JavaScript* + *Frameworks*

Requisitos Funcionais:

- Atualização de conteúdos dinâmica

- Apresentação dos dados provenientes do sistema de monitorização

- Apresentação da alarmística despoletada por eventos no sistema de monitorização

Requisitos Não-Funcionais

- Representação de informação em forma de gráficos e tabelas de acordo com o sistema de monitorização *Zabbix.*

- A alarmística deve-se destacar de entre as restantes informações presentes na página.

- Deve-se usar eficientemente a área de visualização da página para apresentar conteúdos (deve-se evitar grandes espaços em branco).

- Devem-se mostrar tantos gráficos quanto os necessários para obter as informações que no modelo antigo de utilização da interface gráfica *Zabbix* eram obtidas (através de *Screens* criados para o propósito, com conjuntos de gráficos selecionados).

- A informação deve atualizar em períodos convenientes (por exemplo, dados que surjam de monitorização com periocidade muito frequente, devem ser atualizados com maior periocidade do que aqueles que são recebidos menos frequentemente)

- Os gráficos devem atualizar com uma frequência compatível com a velocidade de atualização dos valores que alimentam o gráfico. Desta forma, um gráfico que receba valores de 5 em 5 minutos deve ser atualizado a cada 5 minutos ou menos, para que seja possível acompanhar a evolução dos valores.

- A atualização dos gráficos deve ser, no entanto, ponderada, para que não sejam atualizados com tanta frequência que se torna difícil a sua visualização ou sobrecarreguem o sistema.

Uma vez identificados os requisitos (que acabaram por ser um conjunto pequeno dada a objetividade daquilo que se pretendia ter apresentado no ecrã), passou-se para a fase de desenho do sistema. Este desenho passou por duas etapas: desenho da interface e desenho do sistema.

5.6.1 Desenho do Sistema

Escolha da Linguagem

Para poder começar a desenvolver as funcionalidades necessárias para o novo *dashboard*, foi necessário decidir que opções tecnologias se utilizariam. Fizeram-se levantamentos a nível da utilização da linguagem Java para desenvolvimento de *Websevices*, nomeadamente, a utilização de *Java RESTful Web Services* (JAX-RS) [21]. Esta opção foi ponderada devido ao conhecimento que já existia na utilização da linguagem Java.

As APIs RESTful têm aspetos bastante aliciantes, como a identificação dos recursos via URI e manipulação destes usando apenas as 4 operações de *header* HTTP que o protocolo HTTP apresenta: GET, POST, PUT e DELETE [21]. O facto de ser uma forma leve de transmitir dados (que iriam diretamente por HTTP num formato bem estruturado como o XML ou o JSON) e de as interações com os recursos serem feitas no formato *stateless* (ou seja, as mensagens que são trocadas contêm toda a informação necessária para a execução da operação, sendo a informação passada no URI, em *cookies* ou mesmo escondida em campos invisíveis dos formulários) [21].

Apesar de todas as virtudes, a curva de aprendizagem mostrou-se demasiado grande, devido à inexperiência que se tinha no desenvolvimento de serviços *Web* e de todo o conhecimento necessário para concretizar os servidores de conteúdos (como por exemplo *servlets*). Devido à escassez de tempo que existia para desenvolver a nova interface, decidiu-se abandonar esta abordagem.

A linguagem PHP [22] apresentou-se como uma alternativa igualmente viável para desenvolver o sistema. A sua capacidade de manipular URIs permitiria concretizar o mesmo conceito de "interacção" com os recursos que o JAX-RS apresenta: cada operação a ser levada a cabo, levaria, no URI, toda a informação necessária para a concretizar [22]. É expressamente declarado que há uma transferência de estado ao se passarem parâmetros por URI, no caso do GET, ou por HTTP, no caso de se usar POST [22]. O PHP apresentou uma menor curva de aprendizagem, sendo simples e fácil começar a desenvolver *software*, mesmo tendo em conta o pouco conhecimento que havia sobre o assunto.

Com apenas alguns dias e seguindo alguns tutoriais [22], foi adquirido conhecimento suficiente para começar a implementar o sistema de *dashboard*, tendo-se posteriormente feita investigação, à medida que foi necessário, acerca de técnicas e documentação da linguagem para melhor saber como implementar as ideias para o sistema.

Obtenção da Informação

O problema de como obter os dados de monitorização apresentou-se a dois níveis: onde/como recolher a informação e como guardar a informação.

Para obter a informação, primeiro pensou-se em aceder à base de dados da própria plataforma *Zabbix*. Sendo uma base de dados *MySQL*, seria simples integrar as informações com a interface, acedendo via PHP e apresentando as informações relevantes no *dashboard.* No entanto, a documentação encontrada não era muito explícita quanto ao propósito de cada tabela usada pelo *Zabbix*, e eram demasiadas para se conseguir compreender e formar *queries* SQL para o que se pretendia.

Após alguma investigação descobriu-se que a plataforma *Zabbix* traz, por defeito, uma API [23] para comunicar com a plataforma, oferecendo métodos para obter e até atualizar indicadores e da própria estrutura que o constitui. De acordo com o que se pode ler na documentação oficial, "*a API Zabbix é uma API baseada na Web (...) que usa comandos remotos HTTP para chamar a API. É também uma API JSON-RPC o que significa que todos os comandos enviados para a API devem ser codificados em [formato] JSON*" [23]. Esta API permitiria não só tratar de todos os pedidos necessários

para obter a informação a apresentar no *dashboard*, mas também invalidaria a necessidade de bases de dados adicionais, quer para obter informação quer para a guardar.

Componente Gráfica

Por último, foi necessário decidir como se construiria a interface gráfica da página. A indecisão estava entre usar um formato simples de HTML puro e *JavaScript* para operacionalizar as funcionalidades da página, ou se se queria adicionar alguma *framework* para enriquecer o aspeto visual e operacional da página. Inicialmente foram ponderadas duas *frameworks*: *AngularJS* e *Bootstrap*.

A *framework AngularJS* foi inicialmente ponderada pois, citando a página *Web* da *framework*, o "HTML é ótimo para declarar conteúdos estáticos, mas falha quando se tentam declarar *views* [páginas] dinâmicas para aplicações *Web*. O *AngularJS* permite estender o vocabulário HTML para a aplicação [*Web* a ser construída]." [24]. De facto, esta *framework* parecia resolver o problema do dinamismo das páginas: poder-se-ia ter conteúdo dinâmico em HTML implementando apenas a *framework AngularJS*.

Contudo, e embora todas as virtudes que apresentava, a curva de aprendizagem não foi compatível com o tempo que se tinha para a implementação da nova interface. Mesmo depois de se concluírem os tutoriais, existiam ainda muitas dúvidas de como desenvolver certas funcionalidades com esta *framework* e, devido ao insucesso em desenvolver uma interface usando *AngularJS*, decidiu-se deixar esta ideia para o futuro, onde se poderá repensar a interface usando esta *framework*.

A *framework Bootstrap* [25] é uma *framework* para *front-end*, ou seja, para a página que o cliente visualiza e interage. O objetivo do *front-end* é apresentar informação e recolher entradas que o utilizador quer passar para o sistema, processando-as para depois as usar no desempenho das operações. É, portanto, importante que este *front-end* seja intuitivo e eficiente, sem que se prescinda do aspeto gráfico apelativo. O *Bootstrap* permite tudo isto ao apresentar uma *framework* de CSS e *JavaScript* que oferece um leque vasto de opções para a interface gráfica em HTML, que melhoram a intuição e a clareza das interações com o sistema, oferecendo, ao mesmo tempo, um grafismo limpo e apelativo.

As mais-valias são a quantidade de documentação e a clareza desta, os imensos *demos* que têm para começar a construir uma interface e a simplicidade de implementação (são, maioritariamente, estilos que são acrescentados, não tendo a complexidade programática – semântica – que o *AngularJS* apresentava). O *Bootstrap* apareceu como a ideal conjugação entre apelo visual e simplicidade funcional.

5.6.2 Implementação dos requisitos funcionais e não-funcionais

Requisitos Funcionais

Quanto aos requisitos funcionais, aquele que apresentava o maior desafio era a atualização dinâmica de conteúdos. Como se queria que a informação fosse mantida atualizada sem a intervenção do utilizador, seria necessário um processo automatizado para obter informação do servidor para alterar parcial ou totalmente a página.

Alterar apenas a parte da página que se pretendia atualizar sempre foi o ideal: estruturando bem a página, seria possível atualizar apenas uma parte da estrutura, mantendo o resto da página igual. Como se sabia que existira a possibilidade da informação ser atualizada de forma assíncrona, esta metodologia pareceu a mais indicada.

Atualizar a página toda oferecia, por um lado, simplicidade de implementação (periodicamente recarregava-se totalmente a página), mas retirava o assincronismo que existe entre atualizações de dados, próprios de um sistema de monitorização. Qualquer visualização ou interação com o sistema seria interrompida para a página voltar a carregar.

Decidiu-se, então, que a página seria desenhada tendo em conta divisões estruturais bem definidas, sendo que cada divisão terá o seu conteúdo, atualizado periodicamente (com periocidade programável) e sem influência nas restantes estruturas que compõem a página.

Para satisfazer esta necessidade, a melhor forma seria utilizar pedidos assíncronos ao servidor, sendo o uso de pedidos AJAX (*Asynchronous Javascript and* XML) [26] a melhor tecnologia para o efeito. Esta metodologia permite realizar pedidos ao servidor para atualizar os elementos do DOM (*Document Object Model*) pretendidos, de forma assíncrona e sem interferir no comportamento da página, atualizando informação e efetuando operações em segundo plano [26].

A tarefa de obtenção de dados e alarmística do servidor de monitorização foi extremamente simplificada no momento em que se descobriu a API *Web* do *Zabbix*. Com recurso às operações desta API, recolher informação de monitorização e alarmística para apresentar na página dependia apenas de alguns pedidos ao servidor. O único problema seria como fazer os pedidos ao servidor, uma vez que não se tinha à-vontade com a invocação de RPCs via JSON. Felizmente encontrou-se uma API para o efeito, *PhpZabbixApi* [27], um produto *open*-source, propriedade da C*onfirm IT solutions GmbH* e da autoria de Domi Barton.

Requisitos Não-Funcionais

Os requisitos não-funcionais prendem-se com a forma de apresentar a informação, que neste caso assume uma dimensão tal, que chega a ser tão importante quanto a própria informação: se se falhar na forma de apresentação, não será fácil ou intuitivo perceber o que está a acontecer nos sistemas; se não for objetiva e rápida a compreensão do que se está a observar, pode-se descurar eventuais ocorrências ou mesmo não perceber se se passa ou não algo de errado nos sistemas.

A forma de apresentação é o meio para atingir o fim. Por isso, as decisões de implementação terão sempre de ser fiéis aos requisitos de simplicidade, objetividade, intuição e apelo visual para a interface gráfica. Nas secções seguintes deste capítulo descrevem-se as opções de implementação que respeitam os requisitos não-funcionais, justificando as escolhas para cada página da nova interface.

5.6.3 Desenvolvimento da Página

Inicialmente desenharam-se duas versões de *dashboard*, um simplista, muito orientado à informação e nem tanto ao estilo e grafismo; e outro baseado no *template* de *dashboard* que o *Boostrap* oferece, fazendo uma boa comunhão entre apelo visual e estruturação da página.

A primeira versão foi desenhada a pensar na utilização do espaço da página (maximizar a quantidade de informação a apresentar) e na apresentação de conteúdos em forma de gráficos. Seria um *dashboard* apenas com gráficos e alarmística como principal componente.

Na Figura 35 podemos ver um esboço daquilo que foi pensado para a versão 1 do *dashboard*. Composta por uma estrutura de gráficos, menu de opções e alarmística, esta versão teria gráficos estáticos, a atualizarem periodicamente, e um conjunto de “caixas” com alarmística para as principais abstrações do sistema (servidores de nomes, bases de dados, máquinas *Linux*, máquinas *Windows*, etc.). A barra central teria algumas funções, como mostrar outras informações do sistema que não estavam visíveis.

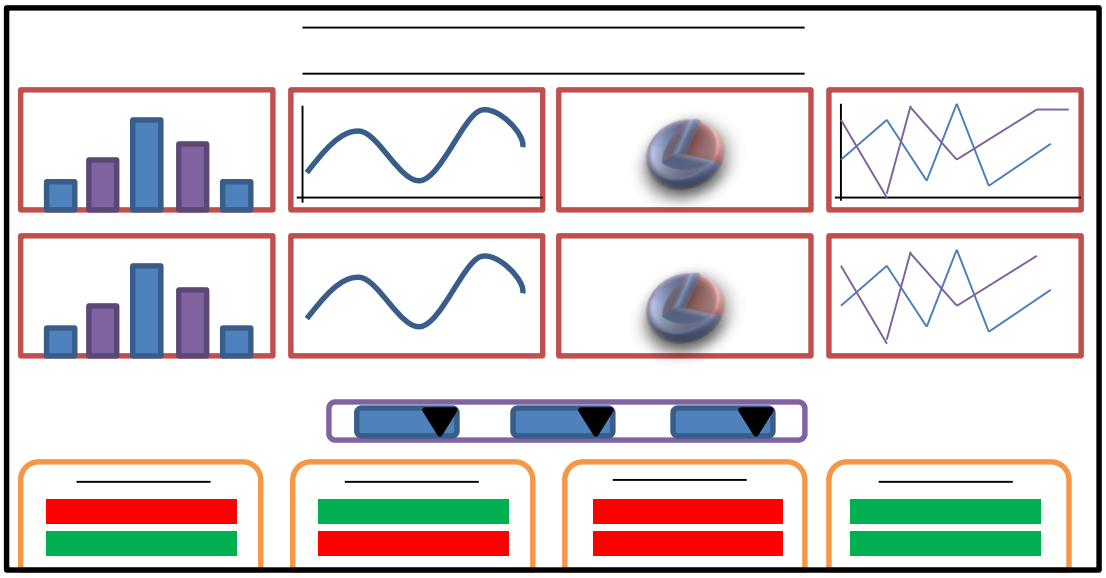


Figura 35 – Esboço da página *Home* da interface *Dashboard*, versão 1

A segunda versão (Figura 36) apresenta uma página mais cuidada e com recurso a técnicas de organização e apresentação de elementos mais complexas, como é o caso do *carousel*, nome dado ao conjunto de imagens que desliza lateralmente, de imagem em imagem (no desenho, representado por caixas com setas nos lados); ou os *collapsable menus*, que ao clicar na seta, expandem uma caixa por baixo, revelando a informação escondida (no desenho, representado na barra lateral esquerda, com setas). A alarmística está presente de uma forma análoga à versão 1, em caixas cujas entradas coloridas indicam a severidade do problema de um dado grupo.

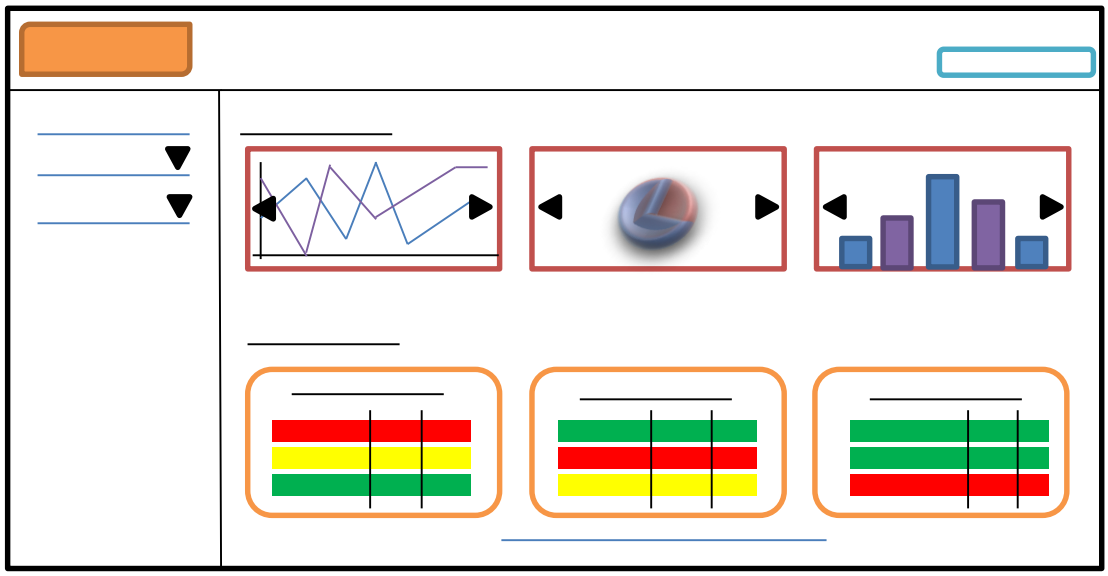


Figura 36 – Esboço da página *Home* da interface *Dashboard*, versão 2

Estes esboços foram apresentados à equipa técnica, sendo depois debatido qual dos dois deveria avançar para implementação. Acabou-se por chegar à conclusão que o ideal seria ter um misto dos dois, na seguinte forma:

- Gráficos: usar-se-ia o *carousel* para mostrar os gráficos, permitindo assim que houvesse mais gráficos no *dashboard* sem com isto implicar páginas longas verticalmente ou encher a página só com gráficos. Acordou-se que existiria um *carousel* para cada sistema (*CPU Load*, número de ligações de rede para cada servidor, etc.) e que as imagens alternariam segundo uma periocidade bem definida.

- Área da página utilizada: acordou-se que a barra lateral no esboço 2 só ocupava espaço que podia ser utilizado para representar informação. Assim, passaram-se as funcionalidades que estavam nessa barra lateral para o cabeçalho, mudando de *collapsable menu* para um estilo de *dropdown menu*.

- Área de Alarmística: o formato da versão 1 seria o desejado, tendo uma caixa para cada sistema a definir, mas apareceriam apenas os problemas identificados para esse grupo. A cada momento, existiria um conjunto de indicações com cores que traduzissem a severidade dos problemas ou a indicação de inexistência de problemas. Com esta alteração, a área de alarmística ficaria mais simples e eficaz na apresentação dos problemas.

Houve outra grande alteração que se decidiu a esta altura: a implementação de um fluxo de navegação na nova interface de *dashboard*.

5.6.4 Fluxo de navegação no dashboard

Desde o início do desenvolvimento do *dashboard* que se questionou se este seria apenas uma página com informação de monitorização e alarmística condensada, ou teria a capacidade de se navegar por páginas para obter mais informação. Concretamente, foi preciso decidir que a nova interface seria apenas uma página com alguns dados de monitorização e alarmística (à semelhança do *dashboard Zabbix*) ou se se pretendia uma interface mais completa, que condensasse num só lugar as capacidades da interface *Web* do *Zabbix*.

O grande problema em ter apenas o *dashboard* para condensar alguma informação, seria a necessidade de aceder à interface *Zabbix* sempre que se quisesse saber algo para além dos problemas e gráficos disponibilizados, como por exemplo, aceder a dados históricos para os vários sistemas e serviços, dar conhecimento de um problema reportado, ou executar alguma ação própria do sistema de monitorização, que só está presente na interface do *Zabbix*. Por outro lado, temia-se cair na tentação de

desenvolver uma nova interface *Zabbix*, tendo tudo o que a outra já tinha, o que seria problemático, considerando não só o tempo e o conhecimento disponíveis para a implementação, mas também a duplicação de informação e operações que ocorreria com esta implementação.

Considerando os cenários atrás descritos, chegou-se a um meio-termo naquilo que o novo *dashboard* seria capaz de fazer. Definiram-se alguns cenários de utilização em que seria útil ter a informação presente no novo *dashboard* e outros em que seria necessário recorrer à interface *Zabbix* para obter mais informação ou realizar alguma operação. Deste modo, criaram-se os seguintes casos de uso:

1) Ver os problemas existentes num dado servidor (*Host*):
   a) Aceder ao *dashboard*;
   b) Se não estiver apresentado na página inicial do *dasboard*, continuar;
   c) Aceder à página do *Host* em questão, navegando na lista de *Hosts*;
   d) Na página de *Hosts*, visualizar a informação em questão.
2) Ver os dados recebidos, na última hora, para certa métrica (*Item*):
   a) Aceder ao *dashboard*;
   b) Se não está representado nos gráficos da página inicial, navegar na lista de *Hosts* e aceder à página do *Host* que contém a informação pretendida;
   c) Na página do *Host*, aceder à lista de *Items* que este disponibiliza, definir a janela temporal desejada e confirmar a operação;
   d) Visualizar a informação pretendida na página.
3) Dar conhecimento de uma ocorrência (*Acknowledge*):
   a) Aceder ao *dashboard*;
   b) Aceder ao problema que se pretende dar conhecimento
      i) Se estiver listado na página inicial, realizar a operação de *Acknowledge* e saltar para o passo *d)*;
      ii) Se não estiver, aceder ao *Host* em questão, navegando pela lista de *Hosts*.
   c) Realizar a operação de *Acknowledge*;
   d) Escrever a mensagem de tomada de conhecimento;
   e) Confirmar a submissão da informação.

Com estes três casos de uso foi possível identificar as principais valências pretendidas para o novo *dashboard*. Tudo aquilo que os principais casos de uso não visaram, seria necessário aceder à interface *Zabbix* para obter. Com esta identificação de necessidades, adicionaram-se os seguintes requisitos:

- Três níveis de organização: Geral (página inicial do *dashboard*), Grupo (página com todos os *Hosts* que compõem um grupo) e Individual (página de cada *Host*). Este fluxo de navegação das páginas, de uma representação macroscópica do sistema até à aos componentes unitários, seria capaz de resolver os problemas de ter duas interfaces

de onde obter informação do estado da infra-estrutura, aparecendo o novo *dashboard* como principal interface de utilização para consulta e mantendo-se a interface do sistema *Zabbix* apenas para usos administrativos ou para consulta de alguma informação mais específica.

- Possibilidade de usar a nova interface para dar conhecimento de um problema: Com esta capacidade passa a ser possível completar a operação de *Acknowledge* usando apenas o novo *dashboard*.

- Capacidade de visualizar a informação histórica de cada *Host*: Com a implementação deste requisito, todos os *Items* que um *Host* tem, terão os seus dados acessíveis via novo *dashboard*. Essa informação estará acessível na página do *Host*, escolhendo-se o *Item* e janela temporal para apresentação dos dados.

Estes novos requisitos resultaram na elaboração da versão 3 do *dashboard* (Figuras 37, 38 e 39).

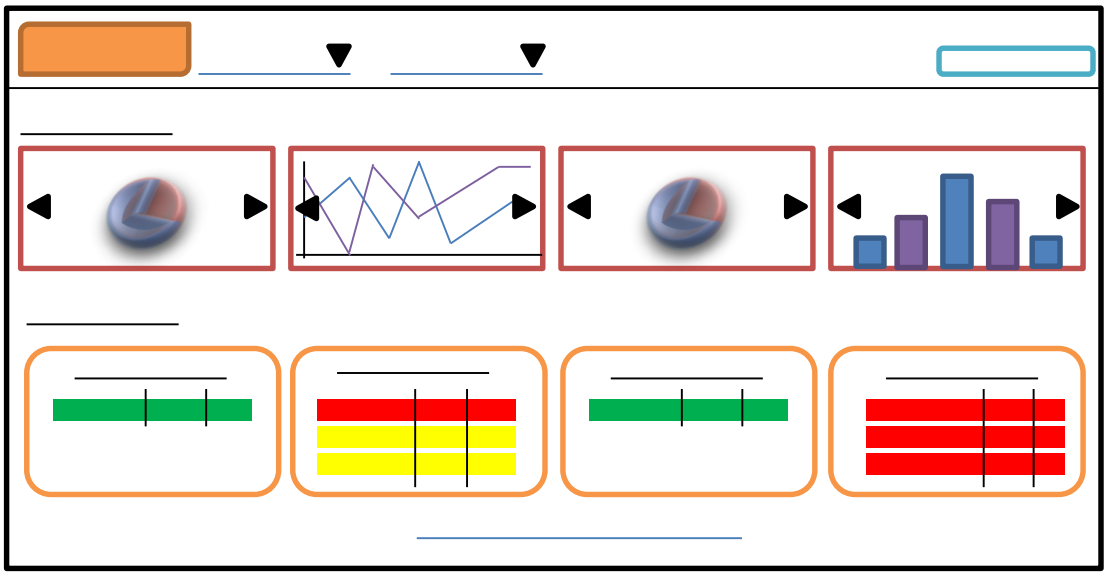


Figura 37 – Esboço para a *Home* do novo *dashboard*, versão 3

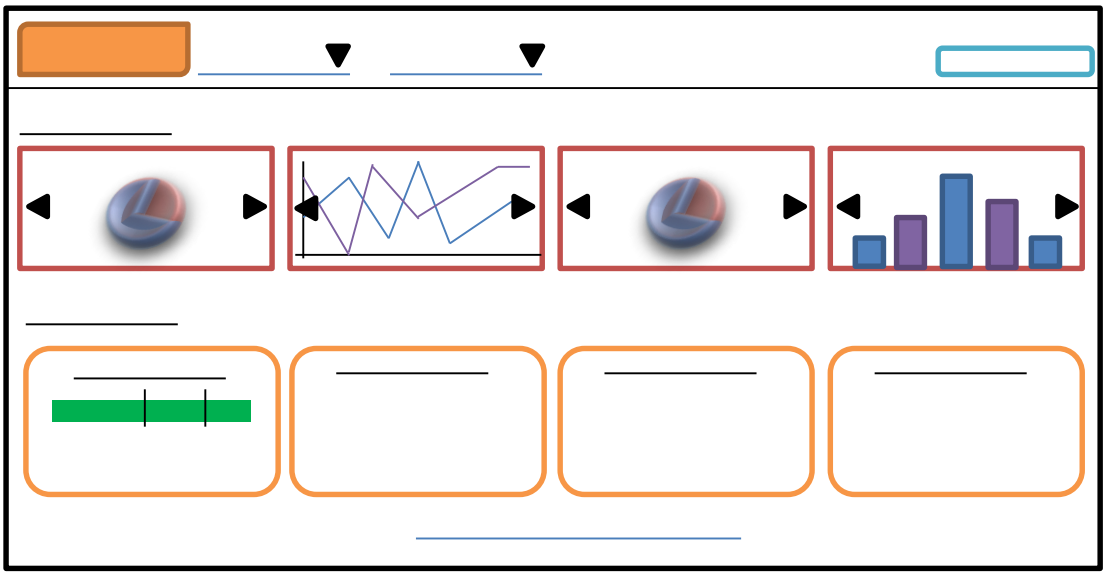


Figura 38 – Esboço da página de *Host* do novo *dashboard* (algumas funcionalidades ainda não estavam presentes por ainda não ter sido pensado quais seriam).

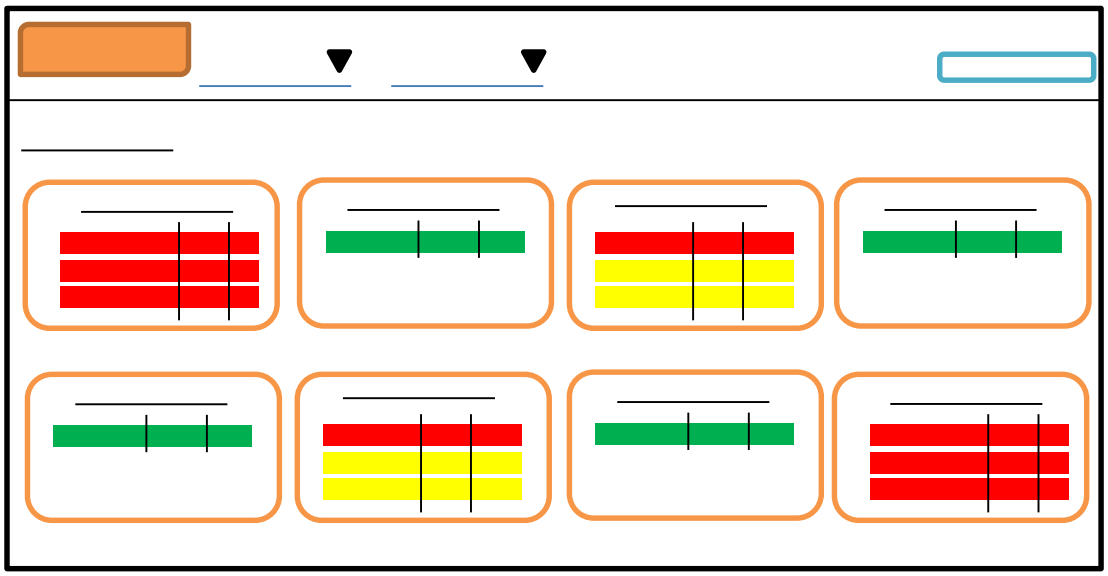


Figura 39 – Esboço da página de *HostGroup*, com as várias caixas de alarmística, uma para cada *Host* que constitui o grupo.

Para a página da funcionalidade de *Acknowledge*, esboçou-se um género de *pop-up*, como exemplificado na Figura 40, onde se introduz a mensagem de tomada de conhecimento, para enviar para o servidor.

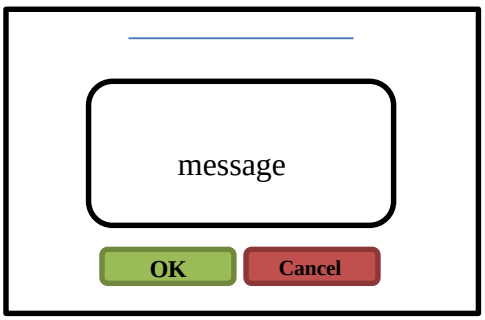


Figura 40 – Esboço para a "página" de *Acknowledge*, onde se dá conhecimento de um evento

5.7 Implementação da nova interface *Dashboard*

5.7.1 Página Home

A página *Home* (Figura 41) é a página principal do *dashboard*. Está desenhada para conter a principal informação acerca do estado da infra-estrutura técnica, apresentando as informações de monitorização atualizadas para os sistemas mais críticos, bem como alarmística para os principais grupos. Esta página é composta por:

- Cabeçalho: O cabeçalho da página contém funcionalidades como os menus com os vários *Hosts* e *HostGroups*, o logo da Associação DNS.PT (que serve também de *link* para voltar à página *Home*) e a funcionalidade de pesquisa.

Este cabeçalho, que está presente transversalmente à interface *dashboard*, contém quatro elementos interativos: o botão *Home* (que está incorporado no nome *Dashboard DNS.PT* e logo da Associação), dois menus expansíveis (*dropdown menus*) com conteúdo dinâmico, e uma caixa de texto para pesquisa, também ela com conteúdo dinâmico.

No caso dos menus expansíveis *HostGroups* e *All Hosts*, a informação para popular a lista vem do servidor de monitorização. Para o efeito, faz-se um pedido ao servidor por todos os *HostsGroups* no sistema e por todos os *Hosts* (que não sejam *Template*) para listar nos menus *HostGroups* e *All Hosts*, respetivamente.

Já no caso da funcionalidade de pesquisa, a abordagem é diferente. Para implementar esta função foi necessário recorrer a pedidos AJAX, a um ficheiro PHP e a um ficheiro XML. O ficheiro PHP operacionaliza a pesquisa por texto que o utilizador insere. É responsável por aceder ao ficheiro XML com todas as entradas que são pesquisáveis (de momento, apenas *Hosts* são pesquisáveis) e recolher a informação que coincide com aquela escrita na caixa de pesquisa. Os pedidos AJAX servem para que

sempre que o utilizador escreva, o PHP pesquise no XML aquilo que o utilizador escreve e integre os resultados como sugestões, por baixo da caixa de pesquisa. Com isto, à medida que um utilizador escreve na caixa de pesquisa, aparece uma lista com itens podem ser clicados, levando o utilizador para a página respetiva.

- Zona de Gráficos: Nesta área da página surgem quatro gráficos. Por defeito, são visíveis os gráficos da quantidade de pedidos por segundo aos servidores, carga de *CPU*, carga de ligações de rede (e estado destas) e estado da JVM das máquinas de *front-end* e *back-end* do sistema SIGA.

Estes quatro conjuntos de gráficos são apresentados num *carousel*, sendo que, periodicamente, os gráficos alternam. Para concretizar isto, recorreu-se a dois ficheiros PHP: um deles, responsável pelos gráficos dos servidores de nomes (que, ao contrário dos outros, não vem do sistema *Zabbix*, mas do sistema DSC), e outro responsável pelos restantes gráficos associados ao estado das componentes do sistema SIGA. O funcionamento destes ficheiros PHP é muito idêntico: cada um operacionaliza a obtenção dos gráficos mais recentes para cada informação pretendida. No caso dos gráficos DSC, a obtenção é feita recorrendo a um pedido com os parâmetros de URL apropriados, sendo que no URL segue a janela temporal de apresentação de dados e o servidor de nomes pretendido. Já no caso dos gráficos do sistema SIGA, o pedido é feito ao sistema *Zabbix*, também por URL, indicando o identificador numérico associado a cada gráfico (estes identificadores estão incluídos no código da página).

Os gráficos são atualizados recorrendo a pedidos AJAX. Para tal, a estrutura da página é divida em *divs*, identificadas por um *id* único, que periodicamente têm o seu conteúdo atualizado com a resposta que vêm dos pedidos AJAX ao ficheiro PHP que trata dos gráficos. Eventuais problemas que se possam ter com *caching* das imagens são resolvidos recorrendo-se à introdução de um *timestamp* (em formato *Epoch*) no URL das imagens.

Para concretizar a atualização automática dos conteúdos, recorre-se a um *script* invocado no *body* da página e define um temporizador (função *setInterval* de *JavaScript*), que periodicamente faz o pedido AJAX que atualiza a informação.

- Zona de Alarmística: Na terceira divisão horizontal de conteúdos da página encontra-se a alarmística. Nesta zona encontram-se os avisos para problemas encontrados para certos sistemas, como servidores de nomes ou o sistema SIGA. Composto por quatro caixas, o resumo da alarmística apresenta as últimas 3 ocorrências de problemas, ordenadas por severidade, e um *link* para as restantes, se existirem.

Nesta caixa (*Thumbnail*, na *framework Bootstrap*) existe uma entrada por cada problema com uma cor que traduz a severidade do problema:

- *Verde*: Não existirem problemas no sistema, mostra um texto acerca disso.

- *Amarelo*: Indica-se que existe um problema de severidade média. Apresenta o nome do *Trigger* despoletado, *Host* que tem problema, a data em que esse problema foi criado e um botão para a ação de *Acknowledge*.

- *Vermelho*: O problema associado é de severidade alta ou muito alta, necessitando de atenção imediata. Tem a mesma estrutura do aviso amarelo.

Como não se quis que a página expandisse demasiado na vertical, todos os problemas que não forem apresentados nesta caixa, estão apresentados na página do *HostGroup* do sistema (Capítulo 5, secção 5.7.2). Adiciona-se uma entrada na lista com o *link* para esta página.

Para apresentar a informação de alarmística, usou-se um ficheiro PHP que obtém todas as ocorrências que estejam com estado de problema, para cada *Host* que integra o grupo. Da lista que é devolvida, filtram-se os problemas com maior severidade para aparecerem primeiro na caixa. Para se manter atualizada a alarmística, usam-se pedidos AJAX para recolher novas informações periodicamente.

A caixa de alarmística apresenta dois outros elementos: um *timestamp* para indicar a última atualização dos dados, e um botão que remete para a página do *HostGroup* respetivo.

Por fim, no rodapé da página *Home*, existe um relógio com a data e hora.

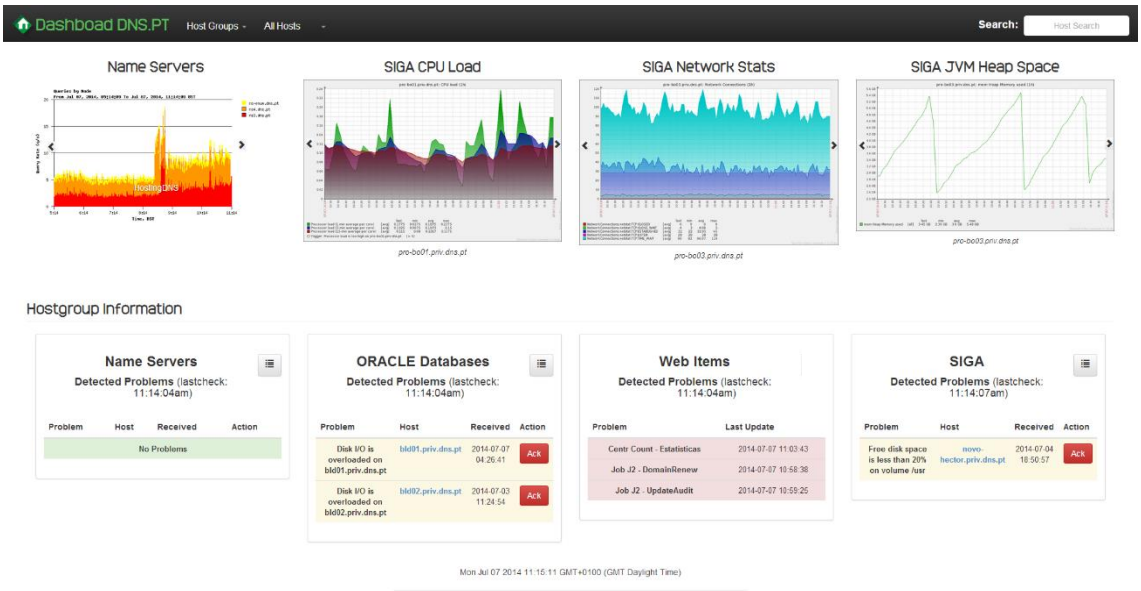


Figura 41 – Versão final da página *Home*, no fim da primeira iteração de desenvolvimento.

5.7.2 Página HostGroups

Esta página serve de intermédio entre a página *Home* e a página de cada *Host*. A página ilustra uma caixa para cada *Host* que faz parte do grupo, e uma entrada para cada problema que esse tenha, com a representação de cores consoante a severidade dos problemas. A Figura 42 ilustra a página no final desta itração.

Em termos estruturais, esta página herda o cabeçalho da página de *Home*. A alarmística apresentada nesta página é muito semelhante à apresentada na página *Home*, sendo as diferenças apenas duas: o número de problemas não está limitado a três e, em vez de existir um botão para ir para a página de *HostGroups*, tem um botão para a página do *Host*.

Os dados para alarmística são carregados a partir de um só pedido AJAX, que é também usado para os atualizar periodicamente, da mesma forma como se faz para a alarmística na página *Home*.

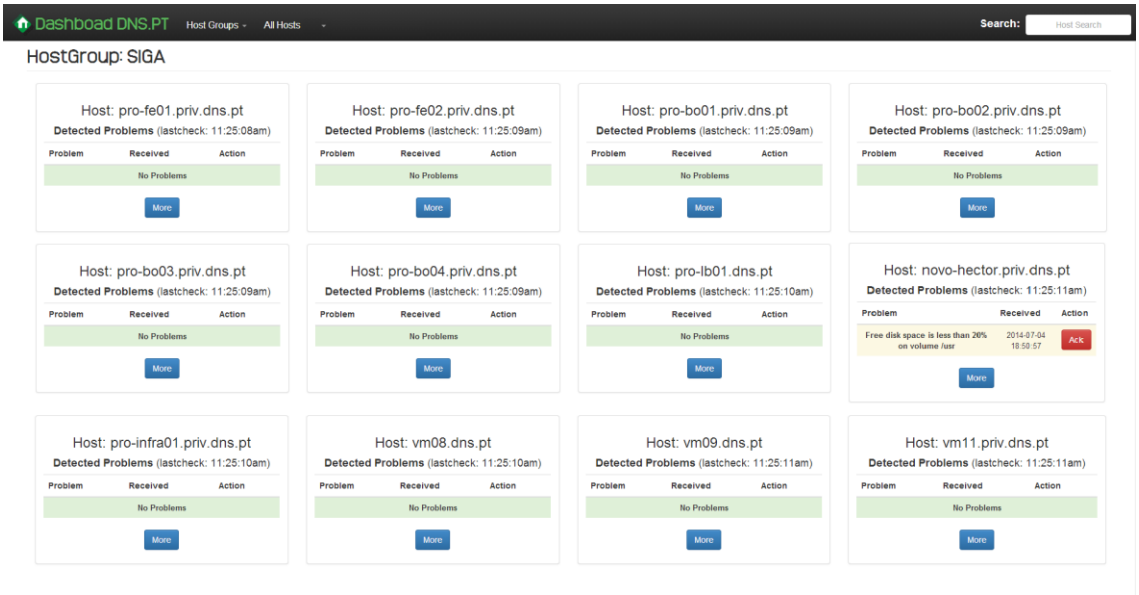


Figura 42 – Versão final da página *HostGroups*, no fim da primeira iteração de desenvolvimento.

5.7.3 Página Host

A página de *Host* (Figura 43) é a página mais específica em termos de visão de sistema. Apresenta informações mais específicas da monitorização do *Host*.

Em termos de estrutura da página, mantém-se o cabeçalho das páginas *Home* e *HostGroup* e fazem-se duas divisões horizontais de conteúdos, conforme acontece na página *Home*. A primeira divisão é a área de gráficos. Os gráficos, transversais a todos *Hosts*, são de estado da utilização da interface de rede, utilização de CPU, de utilização

de disco e estatística de operações de disco. Os gráficos de utilização de disco, por requererem acesso a funcionalidades que não estão presentes em todas as distribuições de *Linux* (necessita de acesso ao */proc/diskstats*), podem não ser apresentados em *Hosts* que não têm *kernel Linux*.

A outra divisão estrutural tem quatro caixas para apresentar quatro conteúdos diferentes: informação do *Host*, alarmística, últimos dados para cada *Item* e histórico de cada *Item*.

A informação do *Host* apresenta dados úteis, como o endereço IP, nome DNS e a distribuição de Sistema Operativo.

A caixa de alarmística é em tudo idêntica à que existe na página *Hostgroups*.

As duas outras caixas têm duas particularidades: não são atualizadas automaticamente e não são carregadas através de um ficheiro PHP, estando codificadas na própria página. Isto prende-se com as características intrínsecas às funções que desempenham, pois para visualizar a informação de um certo *Item*, só se contacta o servidor quando se preenche um formulário.

A caixa do histórico de *Items* (na interface, *Item History*) tem a função de apresentar, quer sob a forma de tabelas, quer sob a forma de gráficos, todos os dados que o sistema de monitorização tem para aquele *Item*, durante um período de tempo definido pelo utilizador no formulário. Existem três elementos a escolher: uma lista de *Item* a visualizar, a janela de tempo para apresentação dos dados, e a forma de apresentação (gráfico ou em tabela).

A apresentção sob a forma de gráfico necessita de um período mínimo de amostragem de uma hora, pelo que se adicionou um botão de informação, com informação de como utilizar o campo da janela temporal.

Quando se submete o pedido, aparece mais abaixo na caixa o gráfico ou a tabela com os valores. No caso do gráfico, aparece ainda um botão para se fazer *zoom*, que maximiza o gráfico (recorrendo-se à funcionalidade de *Modal* da *framework Bootstrap*) e com botões para fechar a janela.

Esta funcionalidade de visualização de histórico tem, no entanto, um problema. Quando se tenta visualizar informação acerca de *Item* cujos valores são palavras, o sistema *Zabbix* é incapaz de apresentar essa informação. Decidiu-se contornar o problema introduzindo uma outra caixa, a *Latest Item Values*, que apresenta os dois últimos valores obtidos para cada *Item* correctamente. A utilização desta função é simples: escolhe-se o *Item* pretendido e submete-se a decisão, aparecendo abaixo os dois últimos valores, sejam eles numéricos ou alfabéticos.

Outro problema que existe com estas duas caixas e com a informação apresentada prende-se com as listagens dos *Items*. As macros e variáveis usadas pelo sistema não são resolvidas no nome do *Item*. Ou seja, alguns *Items* têm no nome *$1*, por exemplo. Após várias tentativas de forçar a resolução das variáveis ou expandi-las, não foi possível faze-lo, sendo mais um problema a ser abordado no futuro, dado o impacto que causa à boa utilização do sistema.

Por fim, tem-se, tal como na página *Home*, em rodapé, um relógio com a data e hora.

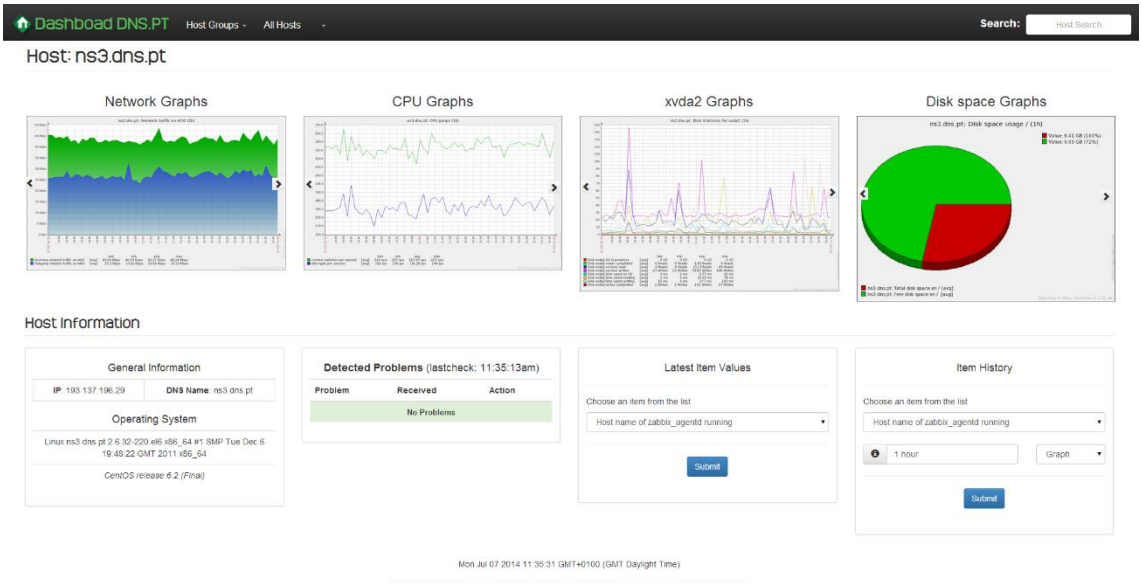


Figura 43 – Versão final da página *Host*, no fim da primeira iteração de desenvolvimento.

5.7.4 Página de Acknowledge

A quarta página da interface diz respeito à função de *acknowledge* – ou tomada de conhecimento – de um problema. Acessível através do botão *Ack* existente por cada problema listado nas caixas de alarmística, a funcionalidade é concretizada numa página unicamente para o efeito. Esta página permite enviar, para o sistema de monitorização, a mensagem de tomada de conhecimento do problema em questão, tendo assim a capacidade de causar alterações no sistema de monitorização, ao invés de apenas o usar para obter informação. A página está ilustrada na Figura 44.

A página tem apenas uma caixa de texto para introduzir a mensagem e dois botões, um para submeter e outro para cancelar. Ao contrário do que se idealizou, o *Acknowledge* não é um *pop-up*, mas como uma página normal, com a mesma estrutura das restantes páginas da interface (tem o cabeçalho igual, exceto as funcionalidades dos *dropdown menus* e da caixa de pesquisa, que foram removidos) e com apenas a possibilidade de submeter a mensagem, ou cancelar e voltar onde estava.

Envia-se a informação que foi introduzida na caixa de mensagem para o servidor sob a forma de um pedido, cuja resposta reflete o sucesso (recebe-se o *ID* do evento que foi dado o conhecimento) ou o insucesso (a resposta não contempla o *ID* do evento em questão). Caso haja sucesso, redireciona-se o utilizador para a página onde estava. Caso contrário, exibe-se um alerta indicando que a tarefa falhou, pelo que deve voltar a tentar, deixando o utilizador na mesma página.

A funcionalidade de redireccionamento desenvolveu-se da seguinte forma: quando se clica no botão de *Ack* em qualquer caixa de alarmística, a página que é carregada leva parâmetros no URL, sendo eles o *id* do evento a dar conhecimento e a página onde se estava anteriormente, para que, quando se decide sair da página de *Acknowledge*, o utilizador seja remetido para a página onde estava. Esta foi a forma encontrada para implementar o redireccionamento, uma vez que o redireccionamento por função *JavaScript* não funcionou (a função *window.history.back(-1)* e outras similares não surtiram o efeito desejado, pelo que foi necessário abordar o problema desta forma).

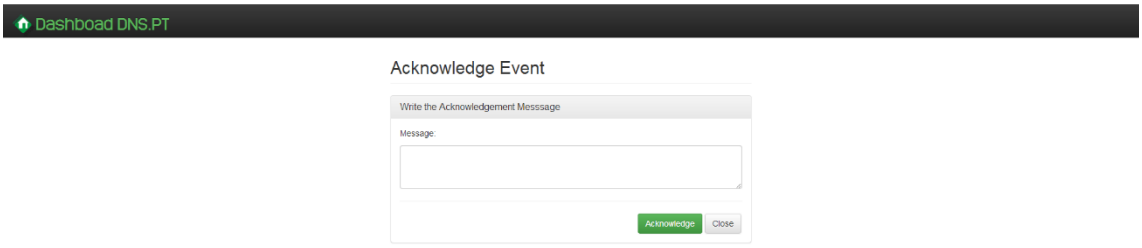


Figura 44 – Versão final da página *Acknowledge*, no fim da primeira iteração de desenvolvimento.

5.8 Avaliação da primeira iteração de desenvolvimento

Depois de criada a primeira versão da nova interface de *dashboard*, foi necessário avaliar se o que tinha sido criado podia ser melhorado. Para tal, submeteu-se a nova interface a testes de aceitação por parte da equipa técnica, tendo sido identificados os seguintes pontos:

Página *Home*

A caixa de alarmística para o grupo *Web Items*, no caso de existirem mais de três problemas, deve apresentar função para visualizar todos os problemas, à semelhança do que acontece coma página de *HostGroups*.

Deve ser possível saber, a qualquer momento, se existem problemas com *Hosts* que não estão a ser apresentados nas caixas de alarmística.

Transversal a todas as páginas

Os botões de navegação sobrepõem-se às imagens, retirando alguma legibilidade aos gráficos. Seria ideal que o controlo de navegação não causasse transtorno na visualização das imagens.

Novas Ideias

Seria importante ter uma página com informação acerca do estado do serviço de resolução de nomes.

5.9 Implementação das oportunidades de melhoria

5.9.1 Página de Web Items

Para poder ter um *link* para todos os problemas dos *Web Items*, foi criada a página dedicada para tal, conforme apresentado na Figura 45. Para além de se mostrarem todos os problemas que existem, achou-se útil apresentar os gráficos associados à velocidade de reposta dos pedidos dos *Web Items*. Para concretizar a página usou-se um novo ficheiro PHP, que apresenta todos os *Web Items* e alarmística associada, bem como os gráficos para o tempo de resposta.

A implementação foi similar à usada para operacionalizar a alarmística de *Web Items* da página *Home*. Para a apresentação dos gráficos, foi necessário fazer mais um pedido, para cada *WebItem*, para obter os *IDs* de cada gráfico, para depois ser possível apresenta-los na página.

Para que esta página não ficasse apenas visualizável a partir da caixa de alarmística na página *Home*, foi adicionado um botão igual ao que existe nas outras caixas e uma entrada para esta página no cabeçalho de todas as páginas, no menu *Custom Screens*.

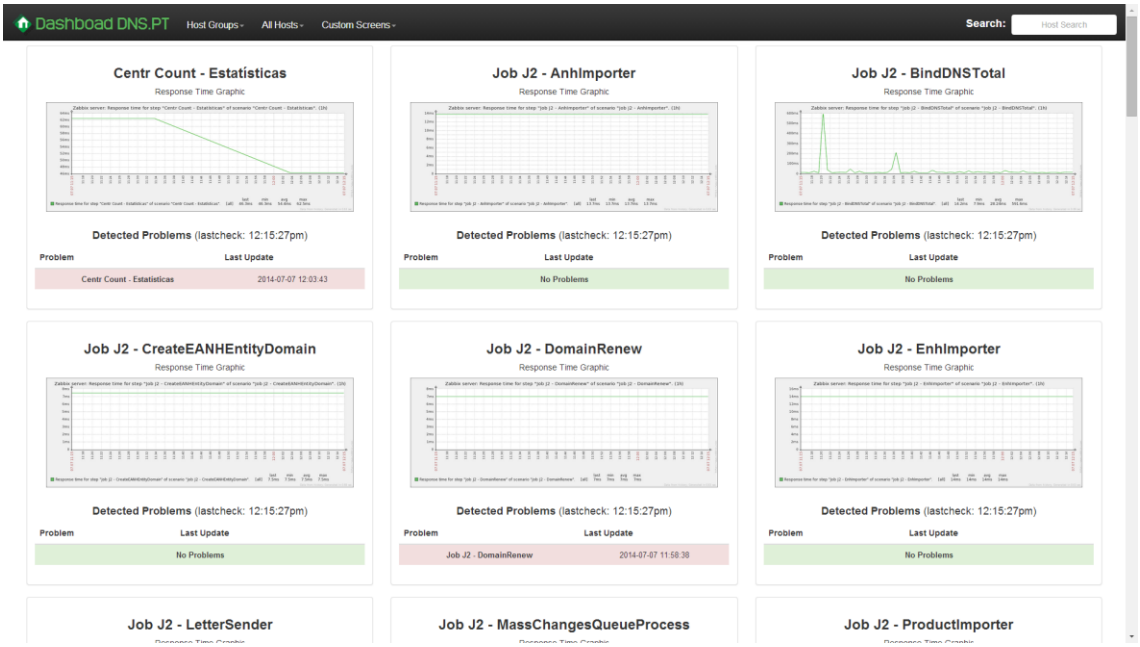


Figura 45 – Página de *Web Items*, com os gráficos e alarmística, no final da segunda iteração de desenvolvimento.

5.9.2 Total de problemas detetados pelo Sistema de Monitorização

A melhor forma encontrada para satisfazer o *trade-off* entre aspeto minimalista e a necessidade de informação, foi colocar, no cabeçalho, onde já existe uma lista de todos os *Hosts*, uma indicação de todos os problemas que existem e que ainda não foram atendidos (que ainda não foi dado *acknowledge*). Desta forma, seria possível ver, em qualquer parte da interface, o número global de problemas e, expandindo a lista (Figura 46), ver para cada *Host*, quantos problemas esse tem.

Para concretizar a parte visual recorreu-se aos *badges* do *Bootstrap*, enquanto a obtenção dos totais e parciais de problemas foram resolvidos fazendo pedidos ao sistema de monitorização. A atualização periódica foi feita via pedidos AJAX.

A implementação da mostragem do número de problemas para cada *Host* e para o total de problemas, foi feita num ficheiro PHP, que faz um pedido ao *Zabbix* por todos os problemas existentes que não foram ainda *acknowldged*. Dessa listagem, contam-se os problemas para cada *Host* e o número total de problemas identificados. Depois criam-se *badges* (que têm a capacidade de representar informação num pequeno circulo

com cor configurável) com o número de problemas para cada *Host* na lista *All Hosts* e um outro *badge* junto ao menu *All Hosts* com o total de problemas.

Os *badges* são de duas cores: verde para quando existe zero problemas e vermelho para quando existe um ou mais problemas.

Todas as páginas atualizam periodicamente os *badges* na listagem do *dropdown menu* e no botão que permite a expansão da lista. Esta funcionalidade é concretizada com recurso a um pedido AJAX, que periodicamente atualiza todos os valores.

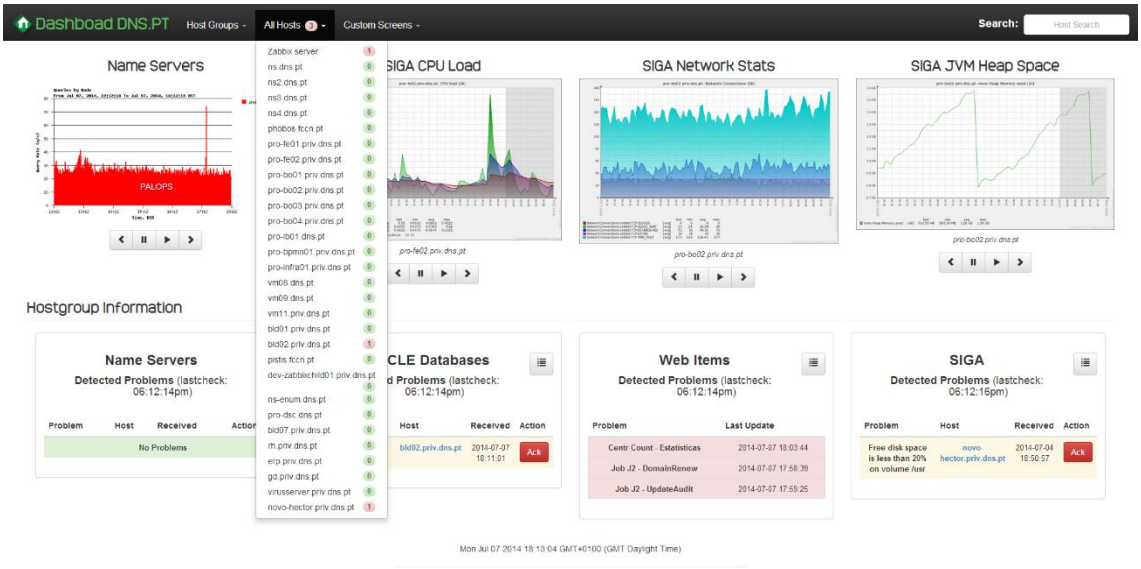


Figura 46 – Página *Home* com a lista do *Hosts* e os *badges* associados, conforme implementado na segunda iteração de desenvolvimento.

5.9.3 Botões de navegação dos carousels

Os botões padrão da componente *carousel* do *Bootstrap* são embutidos na própria imagem que está a rodar. Esta forma, embora faça uma gestão espacial da página mais eficiente, compromete a legibilidade dos gráficos. Portanto, foi necessário conceber uma alternativa que permitisse a navegação sem perturbar a capacidade de visualização dos dados dos gráficos.

Este objetivo foi alcançado com a introdução de quatro botões por baixo de cada *carousel* (Figura 47), tendo botões para cada seta que existia anteriormente, e mais dois botões para duas novas funcionalidades: parar e recomeçar a rotatividade das imagens. A implementação destes botões requereu maior utilização de espaço, mas foi a melhor solução que se conseguiu encontrar. Estes botões manipulam o *carousel* via *JavaScript*, permitindo navegar pelos vário gráficos, parar para visualizar melhor algum deles e retomar assim que se quiser.

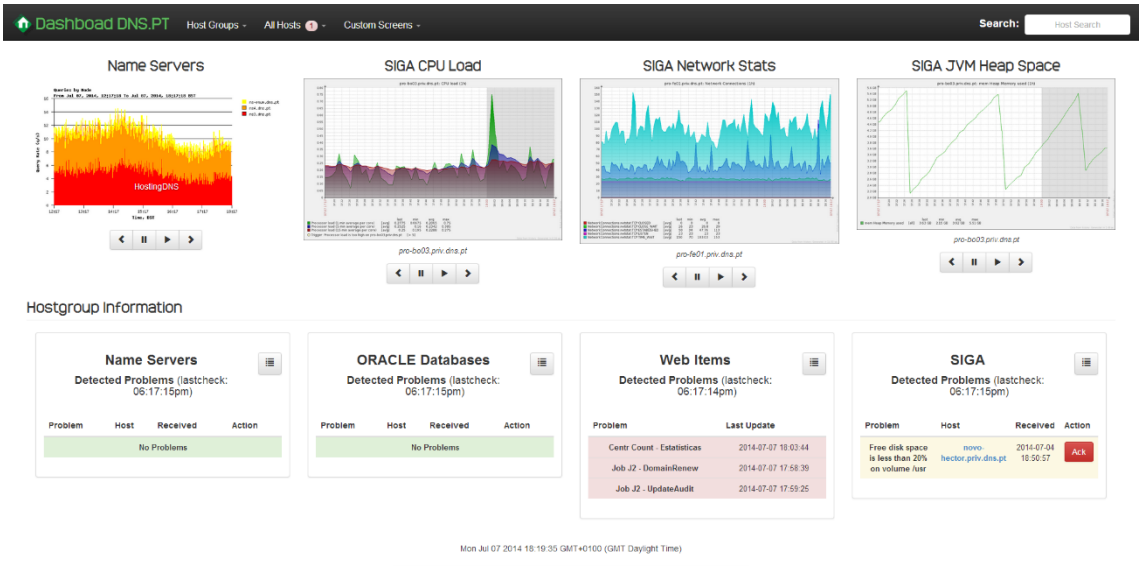


Figura 47 – Página *Home* com os novos controlos de navegação, conforme implementado na segunda iteração de desenvolvimento.

5.9.4 Página de Servidores de Nomes

Esta página já tinha sido considerada antes da implementação do novo *dashboard*. Algumas informações acerca dos Servidores de Nomes são demasiado específicas para tentar generalizar e aplicar numa das outras páginas. Assim foi necessário criar uma página específica, adicionada ao *Custom Screen*, como a págna de *Web Items*.

A estrutura de página é análoga à página *Home*. Na zona de gráficos colocou-se um conjunto de gráficos de estado do serviço *BIND*, um para cada servidor de nomes. Na zona onde se situa a alarmística da página *Home*, configuraram-se quatro caixas, preenchidas com as seguintes informações:

- *Incoming Queries per Second by Server*: Nesta caixa, apresentam-se as médias diárias (desde as 0 horas do dia, até ao momento atual) e último valor recebido de volume por segundo de pedidos ao servidor de nomes, uma entrada por cada servidor.

- *NXDOMAIN Replies per Second by Server*: Esta informação é apresentada numa tabela igual à do item anterior, com o valor médio do dia e o último valor, para cada servidor de nomes. Ter esta informação atualizada periodicamente permite ver se os últimos valores recolhidos estão discrepantes face à média diária, podendo indicar a existência de ataques aos servidores de nomes.

- *IPv4/6 Requests per Second by Server*: Apresenta as médias diárias dos pedidos que chegam em IPv4 e IPv6 a cada um dos servidores de nomes.

- *DNSSEC - Last Signatures*: Esta caixa, referente aos dados de DNSSEC, indica a cada momento se as zonas estão assinadas (ou seja, o *DNSSEC* está funcional para essa zona) ou não. Usam-se cores, verde e vermelho, consoante esteja tudo bem, ou exista algum problema.

Por fim, tem-se, tal como na página *Home* e *Host*, em rodapé, um relógio em tempo real, com a data e hora. A Figura 48 mostra a página no fim desta iteração.

Toda esta informação é atualizada periodicamente com recurso a um pedido AJAX que invoca um ficheiro PHP, responsável por englobar os pedidos e apresentar os resultados.

Figura 48 – Página de *Name Servers* e menu *Custom Items*, conforme implementado na segunda iteração de desenvolvimento.

5.10 Avaliação da segunda iteração de desenvolvimento

Depois de se implementarem todas as alterações e sugestões que surgiram das páginas criadas na primeira iteração de desenvolvimento, voltou-se a avaliar a interface e as novas funcionalidades. Dado que as principais questões ficaram resolvidas nas implementações da segunda iteração de desenvolvimento, decidiu-se traçar objetivos futuros para melhorar a interface, uma vez que o tempo existente não seria suficiente para concretizar tudo o que se pretendia, dada a proximidade do fim do projeto.

Capítulo 6 Discussão

6.1 Análise dos resultados obtidos

6.1.1 Avaliar o antigo sistema

A avaliação do antigo sistema permitu fundar as bases com que se construiu o novo sistema. Foi com base nos levantamentos e na avaliação do antigo sistema que se obtiveram os requisitos para a criação de um sistema que se adaptasse à realidade da Associação. Assim, embora difícil, foi encontrado algum tempo para discutir com a equipa aquilo que se pretendia fazer, tendo a noção que o contributo deles seria sempre uma mais-valia na criação de um sistema que fosse verdadeiramente uma mais-valia para a equipa. Não existiram tantas reuniões quanto se pretendia, o que teve algum impacto naquilo que se conseguiu descobrir (quer numa fase preliminar, quer já aquando da implementação), mas no geral, foi possível conciliar o pouco tempo existente com os objectivos e criar uma boa base para conceber um plano para criar o sistema pretendido.

6.1.2 Desenhar o novo sistema

Um bom desenho só conseguiria ser atingido mediante o levantamento e a avaliação do sistema anterior, com especial ênfase no que estava errado, para se poder solucionar no novo sistema. Todas as dificuldades sentidas na fase de avaliação foram sentidas, transitivamente, nesta fase. Ainda assim, foram feitos os esforços necessários para garantir que se conseguiria criar um sistema que todos os elementos da equipa técnica se identificassem.

O tempo que não foi empregue em reuniões foi usado para investigação de novas funcionalidades a constar no novo sistema, e para avaliar até que ponto as componentes do anterior sistema deveriam passar para o novo.

Avançar em vez de esperar por mais *feedback* dificultou um pouco o desenho do sistema e distanciou-o de um consenso geral, mas possibilitou que o desenho fosse feito de raíz com um conjunto de ideias firmes, sem os inevitáveis desvios que várias

opiniões por vezes trazem. Olhando em retrospectiva reconheço que mais opiniões e sugestões teriam ajudado, mas as avaliações feitas aos desenhos demonstraram que, mesmo assim, as ideias acabaram por coincidir.

6.1.3 Implementar o novo sistema

A implementação do novo sistema foi dividida em duas partes: implementação do sistema de monitorização e alarmística (*Zabbix*) e a criação da nova interface para o sistema (*Dashboard* DNS.PT).

O primeiro foi relativamente mais fácil de concretizar que o segundo. O facto de já haver bastante conhecimento da ferramenta dentro da área técnica facilitou o *debugging* e a resolução dos problemas que foram aparencendo com as configurações, pelo que se excederam as expectativas. A ajuda na instalação dos pré-requisitos técnicos do sistema, bem como o sistema em si, decorreu com supervisão (principalamente do co-orientador Assis Guerreiro) e o sistema *Zabbix* foi configurado rapidamente, sendo que as afinações ao sistema foram feitas no decorrer do mês seguinte. Tendo sido necessárias apenas duas iterações para chegar a uma versão estável do sistema de monitorização, conclui-se que o planeamento e o desenho foram feitos corretamente.

Quanto ao desenvolvimento do *Dashboard*, considero ter sido uma vitória, devido à ambiociosa tarefa que se propôs. Sem ter conhecimentos de PHP nem de *Web design*, contruir uma interface que interligasse com o sistema *Zabbix*, fosse funcional e que esteticamente fosse apelativa foi uma tarefa que não tinha antecipado conseguir com os resultados obtidos. A interface mostra-se capaz de apresentar a informação de forma clara e objetiva, pelo que ajuda nas tarefas do dia-a-dia de monitorização e alarmística.

Tendo aprendido e feito grande parte sozinho (e uma pequena parte com ajuda externa à Associação), considero que o produto final foi bem acolhido e cuja utilização não tem detetado grandes necessidades de melhoramento, para além daquelas que constam do trabalho futuro.

6.1.4 Identificar trabalho futuro

Esta temática é abordada mais à frente neste capítulo (secção 6.4, Trabalho Futuro), mas enquanto objetivo que tinha sido delineado, penso que foi concretizado de forma satisfatória.

A equipa deu o seu *input* e fomentou algumas ideias para o futuro, mas as principais alterações a fazer já foram identificadas desde os planeamentos iniciais.Neste momento a Associação DNS.PT está na iminência de evoluir a sua infra-estrutura técnica e, como consequência, atualizar o sistema de monitorização e alarmística. Independentemente disso foi possível criar a lista de trabalhos futuros conforme se pretendia, dando continuidade aos trabalhos no sistema de monitorização e alarmística.

6.2 Balanço

O balanço a fazer de mais de 9 meses de trabalho foi o de um projeto produtivo, quer para a Associação DNS.PT, quer para a minha realização pessoal e profissional. No final fica um sentimento de objetivo cumprido, dentro do que era possível fazer (de notar que o trabalho feito na Associação DNS.PT não foi todo no âmbito do projeto, nem o sistema de monitorização ficou fechado aquando do término do projeto).

No decorrer dos 9 meses de duração do projeto, foi possível fazer muito daquilo que se queria fazer, mas nem tudo o que foi feito decorreu de acordo com o planeado. Em grande parte, este cenário ocorreu devido aos problemas que houve na coordenação entre a equipa técnica, os tempos necessários para as várias etapas das concretização e, de forma geral, o tempo que se teve para que o sistema fosse feito à medida das necessidades da organização.

Durante a fase inicial do projeto, que culminou com a entrega do relatório preliminar, a Associação DNS.PT estava num ponto de viragem na sua história: houve uma drástica mudança no funcionamento quaotidiano, coisa que se refletiu em mudança de instalações e de várias mudanças a nível da operacionalização do negócio.

Em suma, depois de Janeiro de 2014, houve maior *stress* no seio da Associação, o que levou a que houvesse mais trabalho em curso, novos projetos e, a nível geral, menos tempo para despender nos vários trabalhos que decorriam no âmbito deste projeto. Isso fez com que o pouco tempo que a equipa técnica tinha para auxiliar no projeto diminuísse, traduzindo-se numa maior carga sobre mim para investigar, planear e concretizar todos os aspetos do sistema de monitorização. Ainda assim, saúdo o empenho e dedicação, dentro do possível, que a equipa técnica demonstrou, ao ajudar nas tarefas que, sem essa ajuda, teriam sido impossíveis de concluir.

Portanto, o balanço que se faz ao fim de 9 meses, é um balanço positivo, com obstáculos que foram sendo ultrapassados, uns mais ágil e pacificamente, outros mais problemáticos e custosos, mas que no final potenciaram aprendizagem e me permitiram

crescer enquanto profissional, desenvolvendo capacidades de organização, independência e novas valências técnicas.

6.3 Planeamento

O planeamento que foi traçado no início do projeto, e revisto em Dezembro de 2013, foi parcialmente cumprido. Num cômputo geral, o planeamento foi cumprido, tendo-se conseguido terminar o projeto dentro da data prevista (cerca de dia 16 de Junho de 2014), com as funcionalidades necessárias desenvolvidas e funcionais. No entanto, existiram algumas tarefas do projeto que não foram totalmente cumpridas, como as reuniões com a equipa técnica, tendo-se, muitas vezes, ficado só pela discussão com o orientador Assis Guerreiro, e outras tarefas a serem apenas parcialmente cumpridas, como a fases de testes e a produção de manuais e documentação do sistema.

Os testes foram sendo feitos à medida que era necessário, sem planeamento concreto, apenas definindo um período de tempo para observar os comportamentos e depois agir de acordo com os resultados. Quanto à documentação, grande parte foi criada para compor este relatório, sendo que agora existem alguns documentos individuais que servem de documentação para a ferramenta, até se conseguir criar algo mais formal.

Tirando estas duas fases, conseguiu-se, com alguma destreza, contornar todos os obstáculos que impediam a concretização das tarefas, conseguindo sempre cumpri-las, embora nem sempre nos períodos planeados. Ainda assim, considera-se que o planeamento foi cumprido, embora com algumas falhas, pois conseguiu-se cumprir com o que se prometeu.

6.4 Trabalho Futuro

Esta secção pretende expor as ambições para o futuro, quer de funcionalidades que, ou não ficaram desenvolvidas, ou ficaram apenas parcialemente desenvolvidas durante o tempo do projeto; quer de novas ambições que vêm complementar o trabalho realizado. É uma forma de se dar continuidade ao que foi feito, que em muito se deve ao aspeto mutável e de constante necessidade de adaptação que um sistema de monitorização está exposto no seio de um ambiente corporativo.

Tendo em conta o futuro da Associação DNS.PT e das suas ambições, prevê-se que haja necessidades de adaptação a vários níveis, sendo que o próprio sistema de monitorização terá de se adaptar às novas exigências, sempre que estas surgirem.

6.4.1 Sistema de Monitorização Zabbix

Uma das primeiras alterações que o sistema *Zabbix* irá sentir prende-se com a necessidade de afinar a monitorização das bases de dados. A monitorização e a alarmística serão aconselhadas por um especialista e, portanto, o que existe poderá não ser o que mais se adequa às necessidades reais. Assim que um administrador de bases de dados decidir o que deve e não deve ser monitorizado, o sistema deverá refletir estes conselhos e monitorizar as bases de dados corretamente.

Outra necessidade futura é a afinação geral que os *Items*, transversalmente a todo o sistema. Os *Items* necessitam de ser revistos a nível de periocidade de recolha e de quanto tempo se guardam os indicadores. Embora sempre que se configure um *Item* se tenha atenção a este pormenor, alguns *Items* vieram de *Templates*, outros foram criados à imagem das necessidades da altura e outros foram importados do antigo sistema, sem avaliar se a periocidade era a mais correta.

Este problema toma proporções maiores devido ao armazenamento ser um assunto tão delicado. Otimizar sempre para a poupança máxima deste recurso é algo que tem de se ter sempre em conta, o que impulsiona a necessidade de rever cuidadosamente esta questão.

Num futuro próximo, a Associação DNS.PT prevê alterações na sua infra-estrutura técnica, ponderando-se algumas mudanças a nível do equipamento que serve os vários objetivos da Associação. Assim, é apenas natural que o sistema de monitorização se tenha de adaptar quando tal acontecer, prevendo-se a adição de novas máquinas e, possivelmente, novas tecnologias para concretizar a monitorização dos novos recursos.

Com isto, se as atuais formas de obtenção dos indicadores serão suficientes, ainda não se sabe, mas é certo que o sistema terá de se adaptar à nova realidade, e isso poderá passar por atualizações e implementações de novas tecnologias para se manter atualizada a monitorização e a alarmística.

Devido ao ponto anterior, e ao constante crescimento que se tem verificado para o sistema de monitorização, pondera-se a possibilidade de usar uma das funcionalidades do sistema *Zabbix* para distribuição de carga, o *Zabbix Proxy*. Servindo de ponto intermédio entre o servidor de monitorização e os *Hosts*, esta *proxy* permitirá lidar com a crescente carga que o servidor tem vindo a experienciar, permitindo aliviar um a sua carga. Devido a todas as configurações que terão de ser feitas (quer configurações da aplicação, quer configurações a nível de máquinas para hospedar estas *proxies*), esta funcionalidade será implementada apenas se:

1) O aumento dos recursos de processamento e armazenamento do servidor não apresentarem melhoria no rendimento;

2) Houver máquinas para hospedar as *proxies.*

O ponto 1) é simples de compreender, dado que muitas vezes é mais simples e eficaz dar-se mais recursos a uma máquina do que implementar um sistema de balanceamento de carga.

Como os servidores são virtualizados, os recursos empregues em novas máquinas *proxy* são os mesmos recursos que podem ser dados à máquina que tem o servidor *Zabbix*. Porém, pondera-se migrar o servidor *Zabbix* para uma máquina física dedicada, o que poderá resolver os problemas ou dar mais apoio à ideia de implementar as tais *proxies*. Para tomar uma decisão consciente ter-se-á de fazer um estudo dos prós e contras de aumentar os recursos da máquina *versus* implementar as *Zabbix Proxy*.

Finalmente, e porque era um dos objetivos deste projeto, planeia-se a implementação das sondas DNS que a Associação DNS.PT possui. Tendo sido relegado para um segundo plano – devido à criticidade de outros aspetos – estas sondas apresentam um grande potencial para a monitorização do DNS, principalmente a nível do tráfego. Está-se, neste momento, a planear internamente como abordar a monitorização do sistema DNS usando estas sondas, que virão acrescentar mais qualidades à já vasta capacidade de monitorização que existe para o DNS.

6.4.2 Monitorização Java Mbeans

Toda a implementação do protocolo JMX para monitorização de *MBeans* foi pensada com um propósito simples e concreto: obter informações que as outras vias de monitorização não conseguiam.

Um dos planos para o futuro próximo é integrar a monitorização de mais serviços desta forma, pensando-se, para já, num cenário em que todos os serviços *Web* da Associação têm monitorização através de *MBeans*. Sendo um plano ambicioso (pois são várias dezenas), planeia-se que seja algo a pensar e idealizar durante os restantes seis meses deste ano, sendo que a concretização pode começar antes, mas deverá ficar feita no primeiro trimestre do ano 2015.

No entanto, é apenas um plano que ainda não foi formalmente definido, embora a monitorização dos *MBeans* já tenha dado provas de que é útil para a monitorização destes serviços.

6.4.3 Interface Dashboard

O trabalho futuro aparece como objetivo de médio a longo prazo, para melhorar a interface, quer a nível de *front-end*, quer a nível de *back-end*.

A nível de *front-end*, embora satisfeito com o produto alcançado, existem algumas melhorias, nomeadamente nas páginas dos *Hosts*. Pondera-se uma restruturação dos conteúdos, nomeadamente a nível das caixas com informações sobre *Items*, que deverão apresentar as informações acerca destes sem os nomes das variáveis, pois dificultam bastante o trabalho de encontrar os *Items* pretendidos.

Esta melhoria também implica alterações no *back-end* da página, pois é necessário encontrar uma forma de obter os nomes das variáveis para depois as apresentar nos nomes dos *Items*.

Outra funcionalidade que ficou adiada para futuro desenvolvimento é a capacidade de criar, de forma automatizada, *Custom Screens*, como aconteceu no caso da página de *Name Servers*. Esta sugestão veio da necessidade de visualizar informações acerca de serviços e de servidores que são muito diferentes daquelas que já existem, obrigando a mudar a estrutura da página para algo diferente das outras páginas. Pretende-se ter a capacidade de criar páginas à medida, tendo-se a liberdade de escolher que gráficos e que informações colocar na página.

É certo que a implementação desta funcionalidade será difícil e irá constituir um desafio, não só por tudo o que um processo automatizado de criação de páginas acarreta (ter-se-á de se criar *Templates* para preencher, que de acordo com o que o utilizador escolhe, origine um página à medida), mas também porque o conhecimento de desenvolvimento *Web* e programação em PHP pode não ser o suficiente, sendo necessário despender tempo para aprender todas as técnicas necessárias para desenvolver esta funcionalidade.

Existem outras *frameworks* interessantes para usar na implementação da interface *Dashboard*. O *AngularJS* foi uma das hipóteses contempladas desde o início, mas existem outras que podem ser igualmente úteis e até mais adequadas para o tipo de interface que se pretende alcançar. Por isso, o compromisso futuro será de revisitar outras tecnologias para melhorar o *Dashboard*.

Lista de Acrónimos

AJAX: Asynchronous JavaScript And XML

API: Application Programming Interface

ccTLD: country code Top Level Domain

CGI: Computed-Generated Imagery

CPU: Central Processing Unit

CSS: Cascade Style Sheet

DB: DataBase

DNS OARC: Domain Name System Operations, Analysis, and Research Center

DNS: Domain Name System

DNSSEC: Domain Name System Security Extensions

DOM: Document Object Model

DSC: DNS Statistics Collector

gTLD: generic Top Level Domain

HTML: HyperText Markup Language

HTTP: HyperText Transfer Protocol

IANA: Internet Assigned Numbers Authority

ICMP: Internet Control Message Protocol

ID: Identifier / Identificador

IP: Internet Protocol [address]

IPMI: Intelligent Platform Management Interface

IPv4: Internet Protocol version 4 [address]

IPv6: Internet Protocol version 6 [address]

Java SE: Java Standard Edition

JAX-RS: Java API for RESTful Web Services

JPEG: Joint Photographic Experts Group [Formato Ficheiro de Imagem]

JSON: JavaScript Object Notation

JVM: Java Virtual Machine

MB: Mega Bytes

ODBC: Open DataBase Connectivity

OTRS: Open-source Ticket Request System

PHP: PHP Hypertext Processor

PNG: Portable Network Graphics

RPC: Remote Procedure Call

RTT: Round Trip Time

SIGA: Sistema de Informação e Gestão Administrativa

SMS: Short Message System

SNMP: Simple Network Management Protocol

SOA: Start of Authority

SOAP: Simple Object Access Protocol

SQL: Structured Query Language

SSH: Secure SHell

SSL: Secure Socket Layer

TCP: Transmission-Control Protocol

TLD: Top Level Domain

UDP: User Datagram Protocol

URI: Uniform Resource Identifier

URL: Uniform Resource Locator

XML: eXtensible Markup Language

Bibliografia

[1] ALBITZ, Paul & LIU, Cricket, *DNS and BIND*, 4ª Edição, O'Reilly, Abril de 2001

[2] VERNA, Dinesh Chandra, *Principles of Computer Systems and Network Management*, 1ª Edição, Springer, 2009

[3] Funcionalidades do sistema *Zabbix* – http://www.zabbix.com/functionality.php – consultado a 1 de Janeiro de 2015

[4] Manual do *Zabbix* Versão 2.2 – https://www.zabbix.com/documentation/2.2/manual – consultado a 1 de Janeiro de 2015

[5] Documentação relativa ao ORABBIX – http://www.smartmarmot.com/wiki/index.php/Orabbix – consultado a 1 de Janeiro de 2015

[6] Funcionamento do *software Jenkins* – http://en.wikipedia.org/wiki/Jenkins_(software) – consultado a 1 de Janeiro de 2015

[7] Documentação do *DNS Statistics Collector* – http://dns.measurement-factory.com/tools/dsc/documentation.html – consultado a 1 de Janeiro de 2015

[8] Referencias para instalação do Sistema *Zabbix* – https://www.zabbix.com/documentation/2.2/manual/installation/requirements – consultado a 1 de Janeiro de 2015

[9] Configuração do agente *Zabbix* – https://www.zabbix.com/documentation/2.2/manual/appendix/config/zabbix_agentd – consultado a 1 de Janeiro de 2015

[10] Página do manual de utilização de *UserParameter* – https://www.zabbix.com/documentation/2.2/manual/config/items/userparameters – consultado a 1 de Janeiro de 2015

[11] Especificações do protocolo SNMP – http://www.ietf.org/rfc/rfc1157.txt – consultado a 1 de Janeiro de 2015

[12] Tutorial JMX – http://docs.oracle.com/javase/tutorial/jmx/overview/index.html – consultado a 1 de Janeiro de 2015

[13] Protocolo IPMI – http://en.wikipedia.org/wiki/Intelligent_Platform_Management_Interface – consultado a 1 de Janeiro de 2015

[14] Explicação dos vários tipos de espaços na memória do Java – http://stackoverflow.com/questions/2070791/young-tenured-and-perm-generation – consultado a 1 de Janeiro de 2015

[15] Documentação *DBforBIX* – http://www.smartmarmot.com/wiki/index.php/DBforBIX – consultado a 1 de Janeiro de 2015

[16] Documentação relativa ao conceito *MBeans* – http://docs.oracle.com/javase/tutorial/jmx/mbeans/ – consultado a 1 de Janeiro de 2015

[17] O que é o DNSSEC – http://dnssec.pt/index.php?lang=pt&id=2 – consultado a 1 de Janeiro de 2015

[18] *Monitorização de I/O de Disco usando* Zabbix – http://www.denniskanbier.nl/blog/monitoring/monitoring-disk-io-using-zabbix/ – consultado a 1 de Janeiro de 2015

[19] Princípio do Pareto – http://pt.wikipedia.org/wiki/Princ%C3%ADpio_de_Pareto – consultado a 1 de Janeiro de 2015

[20] WIP: Custom *Dashboard* [da Comunidade *Zabbix*] – https://www.zabbix.com/forum/showthread.php?t=19258 – consultado a 1 de Janeiro de 2015

[21] Documentação da tecnologia JAX-RS – http://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html – consultado a 1 de Janeiro de 2015

[22] Tutorial PHP – http://www.w3schools.com/php/ – consultado a 1 de Janeiro de 2015

[23] Página oficial da *Zabbix* API – https://www.zabbix.com/documentation/2.2/manual/api – consultado a 1 de Janeiro de 2015

[24] Página oficial da *framework AngularJS* – https://angularjs.org/ – consultado a 1 de Janeiro de 2015

[25] Página oficial da *framework Bootstrap* – http://getbootstrap.com/ – consultado a 1 de Janeiro de 2015

[26] Turtorial AJAX – http://www.w3schools.com/ajax/default.asp – consultado a 1 de Janeiro de 2015

[27] Página oficial da API PHP – http://zabbixapi.confirm.ch/ – consultado a 1 de Janeiro de 2015

Anexos

Tabela I - Excerto da Tabela de Levantamento da Monitorização na Associação DNS.PT

Serviço	Monitorização	Descrição	Cód. Amarelo	Cód. Vermelho
Sistema de Monitorização: Zabbix				
Bases de Dados Oracle (prddns1 e prddns2)	Número de Sessões na BD		NA	>= 80%
	Estado da Base de Dados	Se a BD está "viva"	NA	0
	Número de Processos na BD		NA	>= 80%
	Estado da conexão ao Orabix	Se a aplicação de monitorização na base de dados está a enviar dados	NA	< 1
Saúde do SO das máquinas (BackOffices, FrontEnds, Name Servers) - CentOS Máquinas: bld01.priv.dns.pt, bld02.priv.dns.pt, bld03.priv.dns.pt, bld04.priv.dns.pt,bld05.priv.dns.pt, bld06.priv.dns.pt, bld09.priv.dns.pt, bld10.priv.dns.pt,bld11.priv.dns.pt, bld12.priv.dns.pt, dev-bo01.priv.dns.pt, dev-fe01.priv.dns.pt, novo-hector.priv.dns.pt, ns-enum.dns.pt, ns3.dns.pt,ns4.dns.pt, phobos.fccn.pt, pistis.fccn.pt, pro-bo01.priv.dns.pt, pro-bo02.priv.dns.pt, pro-bo03.priv.dns.pt, pro-bo04.priv.dns.pt, pro-bpmn01.priv.dns.pt, pro-fe01.priv.dns.pt, pro-	Modificação no sshd / ssh	O serviço ou o deamon de ligação remota por SSH foi modificado	NA	> 0
	O serviço de SSH está em baixo	não é possivel contactar certo servidor via ssh	NA	0
	Falta de memória RAM livre		NA	< 10 000
	Falta de espaço livre em disco		NA	< = 20%
	Alterado o ficheiro de passwords		NA	> 0
	Alterado o ficheiro de configuração de rede		NA	> 0
	Demasiados processos em execução		NA	> 10
	O host está inacessível	não é possivel contactar certo servidor via mecanismos de rede	NA	2
	O host foi reiniciado	O tempo de vida da sessão é inferior a um certo valor, que indica que foi reiniciado	NA	< 600
	A carga do processador	Verifica se a carga está muito alta	NA	> 5

Group	Host	Linux	FreeBSD	Windows	BIND	JBOSS	LIFERAY	Base de Dados	Mail Server
Zabbix-Servers	dev-zabbixserver01.priv	●							
	dev-zabbixchild01.priv	●							
	dev-zabbixchild02.priv	●							
	vm18.priv.dns.pt	●							
dev	dev-intra.priv	●							
Name Servers	ns.dns.pt		●		●				
	ns2.dns.pt		●		●				
	ns3.dns.pt	●			●				
	ns4.dns.pt	●			●				
	ns-enum.dns.pt	●			●				
	phobos.fccn.pt	●			●				
	ns.dns.br	X	X	X	● (DSC)				
	ns-pt.nlnetlabs.nl	X	X	X	● (DSC)				
SIGA	*qua-bo1*	?				?	?		
	qua-bo2	?				?	?		
	qua-bo3	?				?	?		
	qua-fe01	?				?	?		
	qua-bpmn01	?				?	?		
	qua-infra01	?				?	?		
	pro-lb01	●							
	pro-bo01.priv	●					●		
	pro-bo02.priv	●				●			
	pro-bo03.priv	●				●			
	pro-bo04.priv	●				●			
	pro-fe01.priv	●					●		
	pro-fe02.priv	●					●		
	pro-bpmn01.priv	●							
	pro-infra01.priv	●						● MySQL	
	vm08.dns.pt	●						● MySQL	
	vm09.dns.pt	●							
	vm11.priv	●							
	novo-hector.priv	●							
BD	bld01.priv	●						● ORACLE	
	bld02.priv	●						● ORACLE	
MAIL	pistis.fccn.pt	●							●
Interno	vmwim.priv			●					
	erp.priv			●					
	rh.priv			●					
	virusserver.priv			●					
	gd.priv			●					

Tabela II – Levantamento das máquinas e serviços a incluir no novo sistema de monitorização.

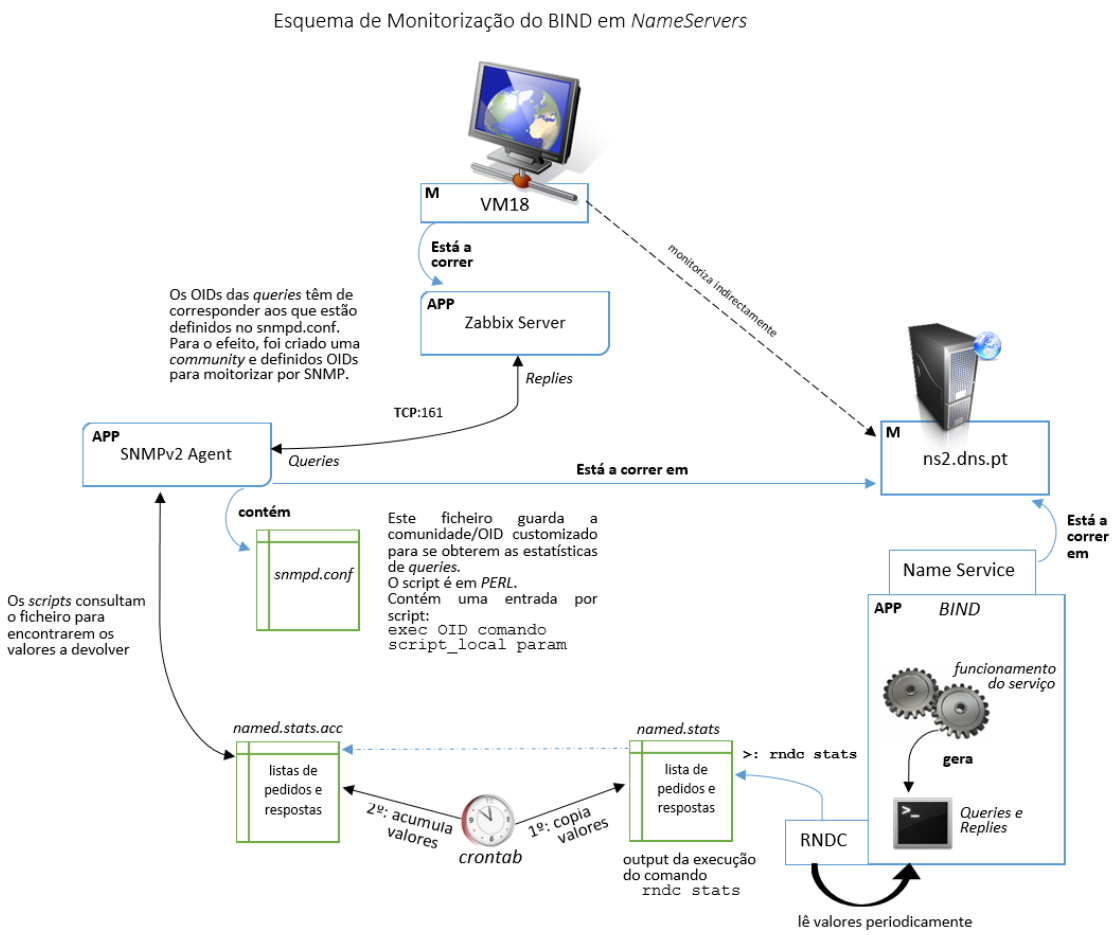


Figura I – Esquema de monitorização do *BIND* em *Name Servers*

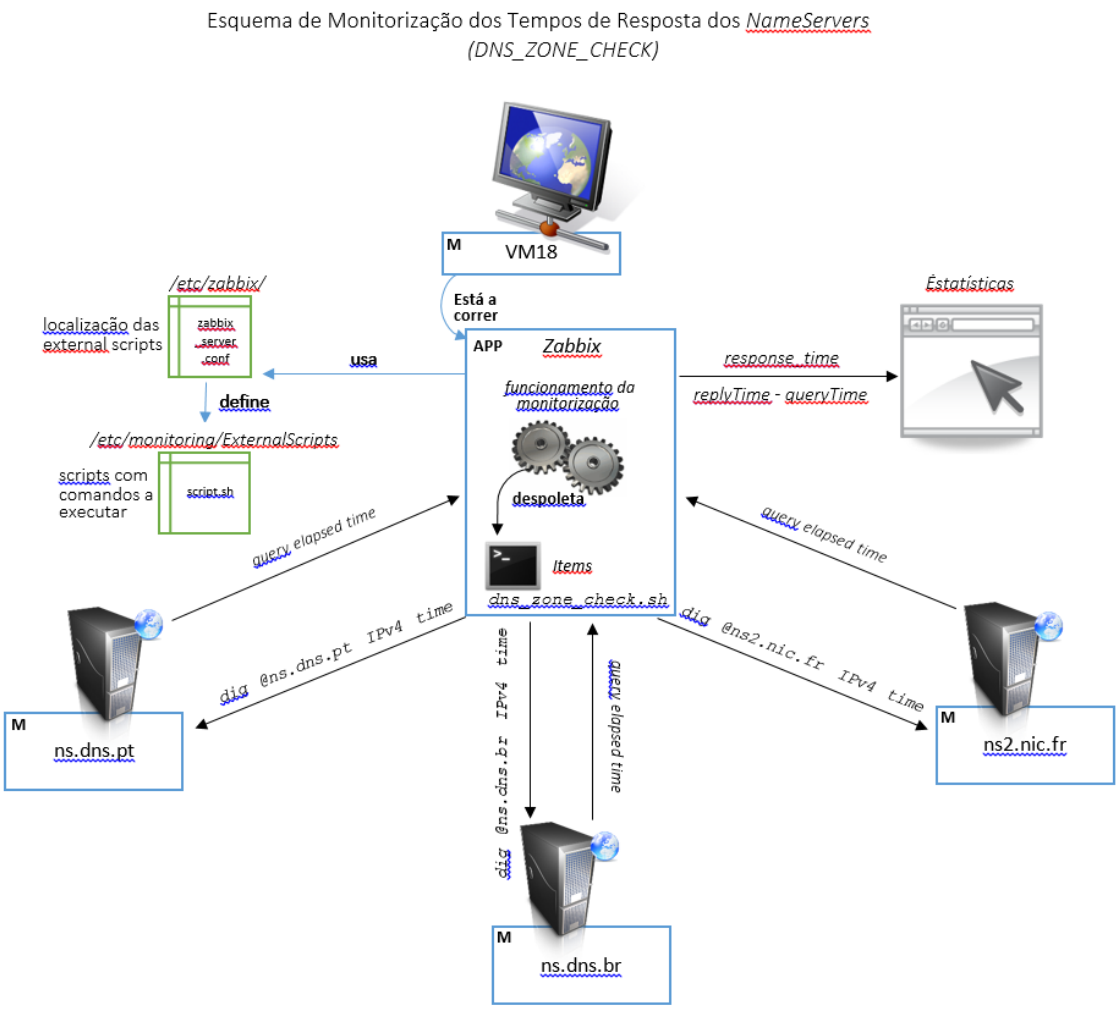


Figura II - Esquema de Monitorização dos Tempos de Resposta dos *Name Servers*

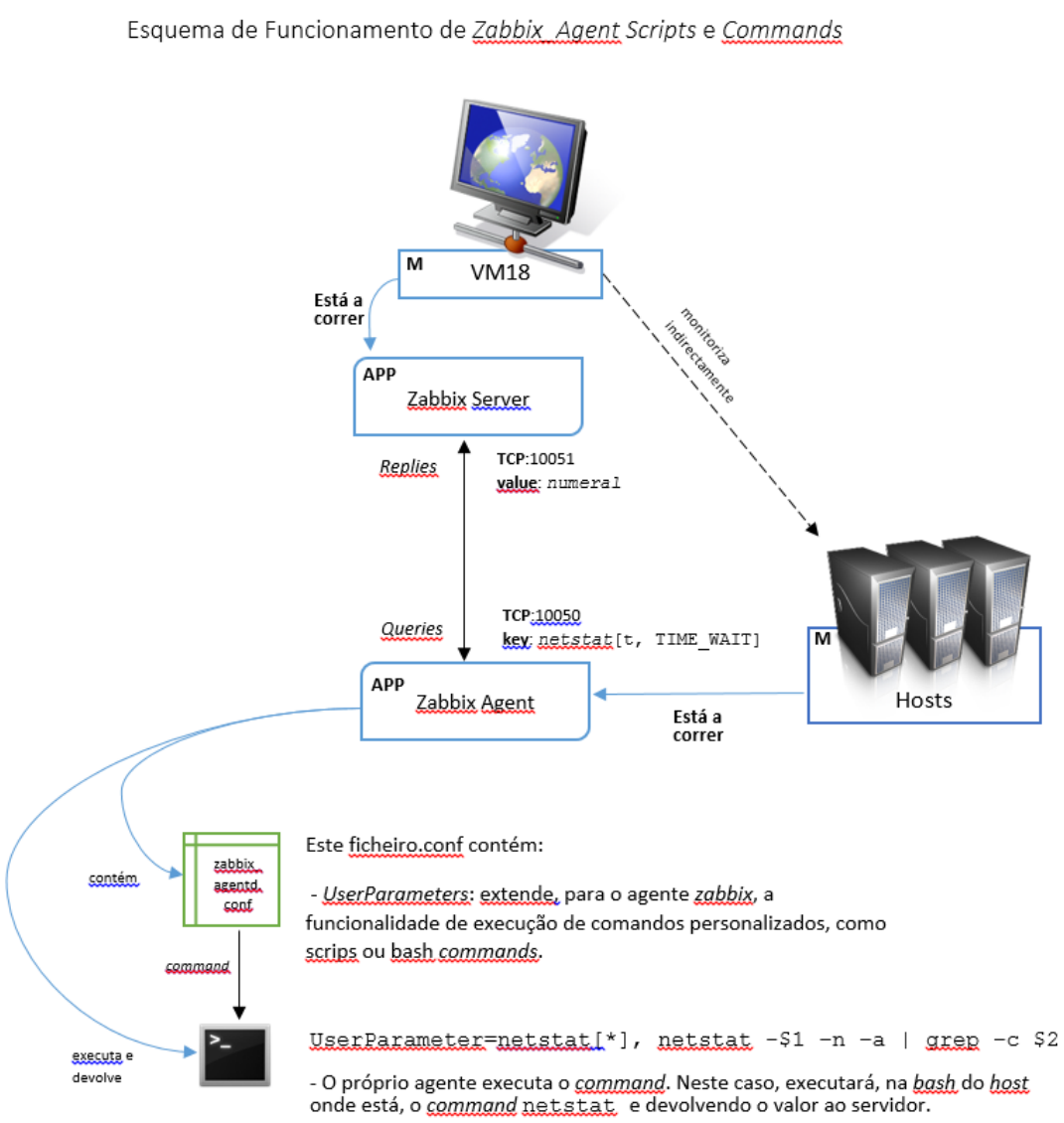


Figura III - Esquema do Funcionamento do NETSTAT, baseado em Zabbix_Agents

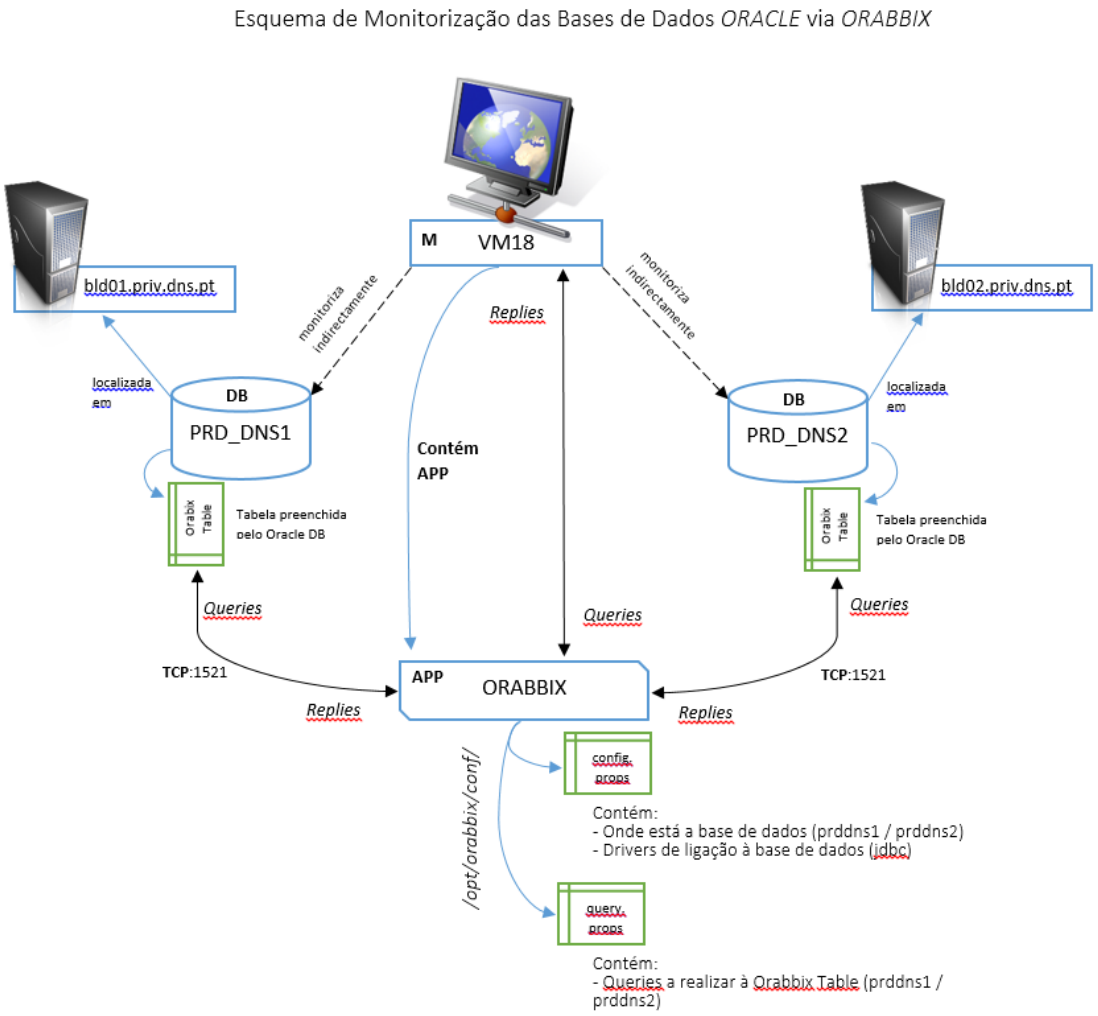


Figura IV - Esquema de Funcionamento da Monitorização das bases de dados usando *ORABBIX*

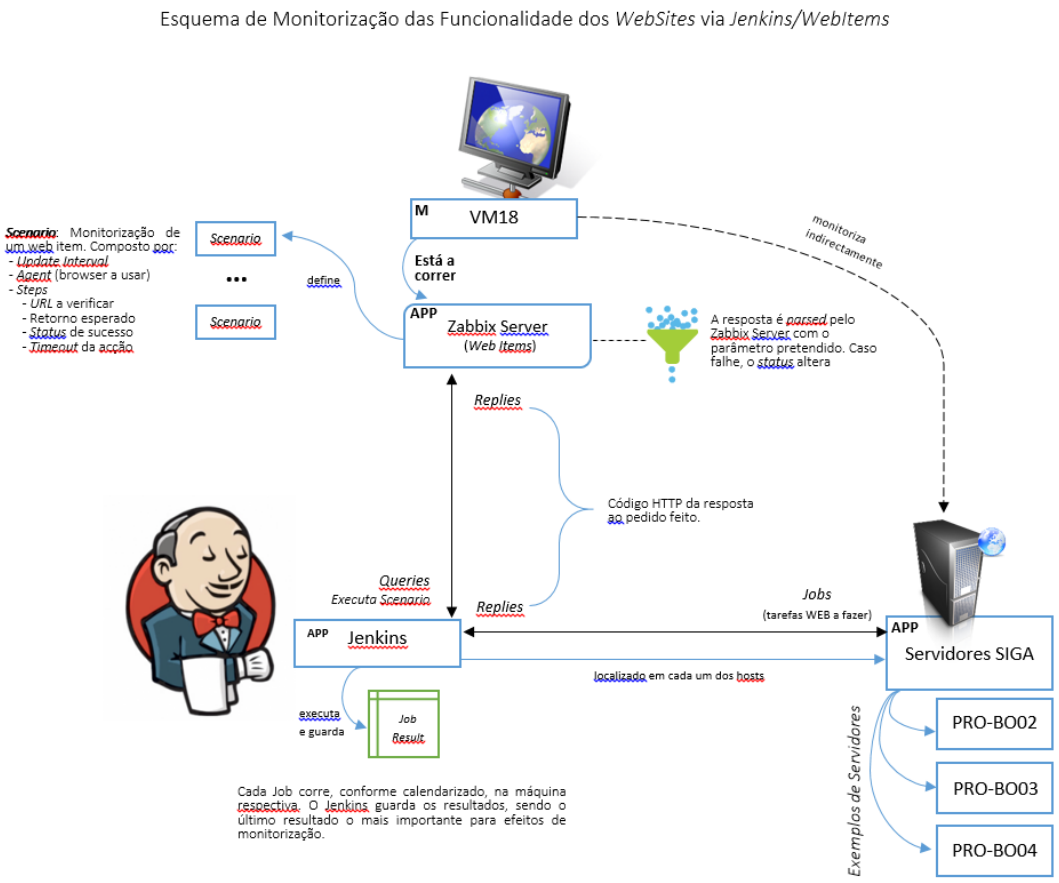


Figura V - Esquema do Funcionamento da Monitorização dos *Websites*, usando aplicações externas (*Jenkins*) e internas (Web *Items* do *Zabbix*)